

# **SDK™-2920 SYSTEM DESIGN KIT USER'S GUIDE**

Order Number: 162418-002

Copyright © 1981 Intel Corporation  
Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department  
Intel Corporation  
3065 Bowers Avenue  
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP  
CREDIT  
i  
ICE  
iCS  
im  
Insite

Intel  
intel  
Intelevison  
Intellec  
iRMX  
iSBC  
iSBX

Library Manager  
MCS  
Megachassis  
Micromainframe  
Micromap  
Multibus  
Multimodule

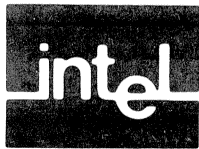
Plug-A-Bubble  
PROMPT  
RMX/80  
System 2000  
UPI

A461/1081/3K SVP

REV.	REVISION HISTORY	DATE
-001	Original issue.	2/81
-002	Minor technical changes.	10/81







## PREFACE

This is the user's guide for the SDK-2920 Design Kit. Included within this guide are descriptions of 1) ROM resident monitor, 2) development board electronics and 3) applications board usage. Before using this manual you should be familiar with 2920 operation and program development techniques. Additional information is available in the following Intel publications:

- *SDK-2920 System Design Kit Assembly Manual*, Order No. 162421
- *2920 Assembly Language Manual*, Order No. 9800987
- *2920 Simulator User's Guide*, Order No. 9800988
- *2920 Digital Signal Processor Applications Handbook*



## SERVICE ASSISTANCE

If, following assembly, you cannot get your kit to operate satisfactorily, the Intel Technical Support Center "Service Hotline" is available for assistance. This service is provided during the hours of 8:00 A.M. to 5:00 P.M. (Mountain Time), Monday through Friday. The toll-free Hotline telephone numbers are:

(800) 528-0595 when dialing outside Alaska, Arizona and Hawaii

(602) 869-4600 when dialing from Alaska, Arizona or Hawaii

The Hotline is intended expressly to help you get your kit running and is *not* intended to assist you in circuit designs or applications. Telephone assistance is limited to *one* call per problem. If a problem cannot be remedied over the telephone, you may, at your discretion, return your assembled kit to Intel for repair. To return your kit, a Return Authorization Number *must be obtained* from the Technical Support Center prior to sending in your kit. Also, either a purchase order number for the repairs must be furnished to the center or a money order (no personal checks please) must be included with the kit being returned. Repairs resulting from defective components supplied with your kit will be done at no charge, and all prepayments will be refunded. Repairs necessitated as a result of customer error, damage or misuse will be billed at a fixed, flat-rate charge which will be quoted by the Technical Support Center.

### NOTE

The Technical Support Center will not repair an SDK-2920 Kit that has been modified and, when circuitry has been added to the user design area, may request that the circuitry be disconnected prior to submitting the kit to the center for repair.





# CONTENTS

<b>CHAPTER 1</b>	
<b>GENERAL INFORMATION</b>	<b>PAGE</b>
Introduction .....	1-1
Uses for the SDK-2920 Design Kit .....	1-1
Compatibility with Design Aids .....	1-1
Specifications .....	1-2

<b>CHAPTER 2</b>	
<b>PHYSICAL DESCRIPTION</b>	
Introduction .....	2-1
Development Side .....	2-1
Applications Side Physical Description .....	2-3

<b>CHAPTER 3</b>	
<b>KEYPAD MONITOR</b>	
Introduction .....	3-1
Monitor Power Up .....	3-2
RESET .....	3-3
Editing Programs .....	3-3
Loading Programs .....	3-9
Converting Number Bases .....	3-19

<b>CHAPTER 4</b>	
<b>INSTRUCTION SET</b>	
How To Use This Chapter .....	4-1
Sequence Field .....	4-1
ALU Field .....	4-1
Destination Field .....	4-4
Source Field .....	4-5
Shift Field .....	4-5
Analog Field .....	4-6

<b>CHAPTER 5</b>	
<b>PERIPHERAL INTERFACES</b>	
Introduction .....	5-1
Serial Transfer Port .....	5-1
Development System Interface Connections .....	5-2
Printer Interface Connection .....	5-3
Object Code Output Format .....	5-4
Cassette Interface .....	5-5

<b>CHAPTER 6</b>	
<b>APPLICATIONS SIDE</b>	<b>PAGE</b>
Example of 2920 Application .....	6-1
Power Supplies .....	6-2
2920 Clock Inputs .....	6-2
Reference Voltage .....	6-4
Digital Outputs .....	6-4
Access to 2920 I/O Pins .....	6-4
Input and Output Circuit Considerations .....	6-5
Output Circuit Considerations .....	6-7
Sample Program .....	6-7
Sample Program Listing .....	6-8
2920 Socketing Procedure .....	6-9

## **APPENDIX A**

### **POWER REQUIREMENTS**

## **APPENDIX B**

### **COMMAND SUMMARY**

## **APPENDIX C**

### **ERROR MESSAGES**

## **APPENDIX D**

### **TWO'S COMPLEMENT DATA HANDLING IN THE 2920**

## **APPENDIX E**

### **DISCUSSION OF CARRY AND OVERFLOW CONDITIONS**

## **APPENDIX F**

### **BIT PATTERNS OF THE 2920 ASSEMBLY LANGUAGE MNEMONICS**

## **APPENDIX G**

### **HEXADECIMAL/BINARY CONVERSION TABLE**



## TABLES

TABLE	TITLE	PAGE
1-1	Specifications .....	1-2
3-1	Conversion Limits .....	3-20
4-1	Constant Codes .....	4-5
4-2	Scaler Codes .....	4-6
6-1	Power Requirements .....	6-2
6-2	Maximum Clock Frequencies .....	6-3



## ILLUSTRATIONS

FIGURE	TITLE	PAGE	FIGURE	TITLE	PAGE
2-1	Development Side .....	2-1	5-3	Baud Rate Selection Jumpers .....	5-2
2-2	Key Switch Arrangement .....	2-2	5-4	Typical Output File in Hexadecimal Format .....	5-4
2-3	Display Format on LED Filter .....	2-2	5-5	Cassette Interface Hookup Locations ...	5-5
2-4	Applications Side .....	2-3	6-1	Applications Side .....	6-1
3-1	Monitor Structure .....	3-1	6-2	Clock Inputs .....	6-3
3-2	Command Interactions with Instruction Memory .....	3-2	6-3	2920 Jumper Pins .....	6-5
3-3	Instruction Fields and Legal Entries ...	3-5	6-4	Aliasing Considerations .....	6-7
3-4	Display Panel .....	3-6	6-5	Sample Program Input and Output Waveforms .....	6-8
3-5	LOAD Command and Display Tree ....	3-9			
5-1	Peripheral Interface Locations .....	5-1			
5-2	RS-232 and Current Loop Pin Assignments .....	5-2			



# CHAPTER 1

## GENERAL INFORMATION

### Introduction

This is the user's guide for the SDK-2920 Design Kit. Now that you have completed the assembly and initial checkout of your SDK-2920, this guide will provide insight into the operation and function of the kit and also will explore applications that can be implemented either off-board or within the user design area.

### Uses for the SDK-2920 Design Kit

Capabilities of the kit include breadboarding applications, assembling 2920 programs, editing programs, programming the 2920 EPROM, communication with development systems, cassette storage of programs, listing to a printer, and number conversion.

**Breadboarding.** The breadboard is used to develop circuits for evaluation or for prototype applications. Components are supplied for several real-time applications.

**Assembling and Editing.** The edit feature includes an assembler, disassembler, hexadecimal display, symbolic 2920 instruction display, and single keystroke entry of many 2920 instructions fields.

**2920 EPROM Programming.** The development board includes a zero insertion force socket plus the necessary controls to program the 2920 EPROM. In addition, before programming the EPROM, the SDK-2920 Design Kit checks to verify correct socketing of the 2920 and displays warnings if your program contains possible error conditions.

**Communications.** The SDK-2920 Design Kit interfaces with Intel Development Systems to pass 2920 object and source programs. Cassette storage capability for 2920 programs is provided as are provisions for printing 2920 programs on local terminals or printers.

**Number Conversion.** The kit provides a number conversion program to convert between binary and decimal.

### Compatibility With Design Aids

Applications software for the 2920 can be designed and debugged by using an Intellec Microcomputer Development System, the 2920 assembler, software simulator, SPAS-20 compiler, and a UPP 102 or 103 with an EPROM personality card and adapter socket. Transfer of 2920 software between the SDK-2920 Design Kit and the development system is easily accomplished.

## Specifications

Specifications for the SDK-2920 Design Kit are shown in Table 1-1.

**Table 1-1. Specifications**

<b>CENTRAL PROCESSOR</b>	
CPU:	8085
Clock Frequency:	6.144 MHz
<b>MEMORY TYPE</b>	
ROM	6K bytes 2732/2716
RAM	1K bytes
<b>MEMORY ADDRESSING</b>	
ROM	0 through 17FFH
RAM	2000H through 23FFH
<b>INPUT/OUTPUT</b>	
Serial RS-232 or user configurable current loop, transfer rate selectable from 110 to 9600 baud Cassette interface	
<b>PHYSICAL CHARACTERISTICS</b>	
Length	16 in.
Width	10 in.
Height	1.75 in. (excluding display)
<b>POWER REQUIREMENTS (See Appendix A)</b>	
V <sub>cc</sub>	+5 volts for development and applications circuitry
V <sub>i</sub>	+ 12 volts for PROM programming and RS-232 interface
V <sub>ap1</sub>	-12 volts for RS-232 interface
V <sub>ap2</sub>	-5 volts for applications circuitry
<b>ENVIRONMENT</b>	
0 to 50°C operating temperature	

### Introduction

The SDK-2920 Design Kit is physically divided into two major areas: the development side and the applications side. The function of the development board is to create and modify 2920 programs, plus program the 2920 EPROM. The function of the applications board is to provide a convenient prototype area with some basic circuits in place.

### Development Side

The development section consists of key switches, alphanumeric display, serial port, cassette interface, EPROM programmer, and control monitor (Figure 2-1).

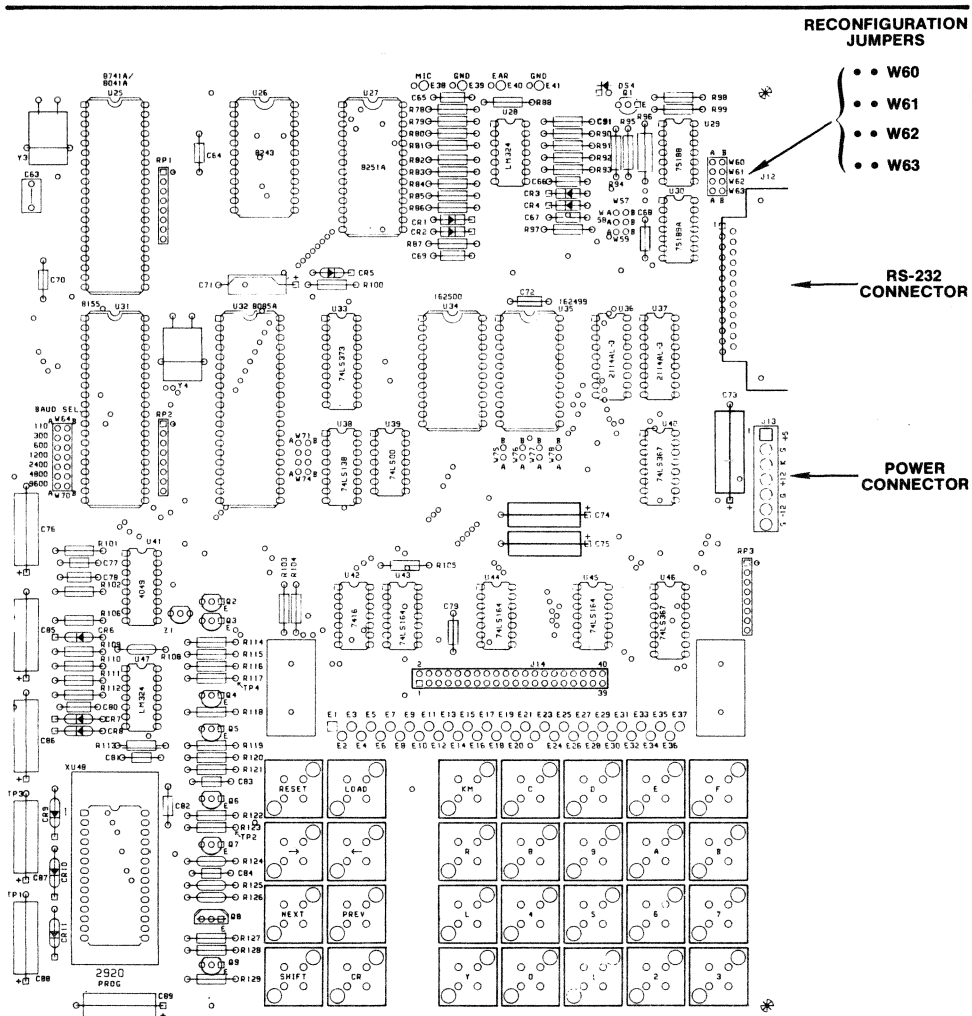


Figure 2-1. Development Side

0226

## Key Switches

The key arrangement, shown in Figure 2-2, consists of 28 keys separated into two groups, a group of eight control keys on the left side and a group of twenty data keys to the right of the control keys. The shift key selects the upper function on the keys.

CONTROL SECTION		2920 DATA KEYS				
RESET	LIST LOAD	KP KM	ADD C	ABS D	ABA E	AND F
HEX/ASM →	EDIT ←	+/- R	SUB 8	LDA 9	LIM A	XOR B
INSRT NEXT	DEL PREV	L ●	IN 4	OUT 5	CVTS 6	CVT 7
SHIFT	CONV CR	DAR Y	NOP 0	CNDS 1	CND 2	EOP 3

Figure 2-2. Key Switch Arrangement

0227

## Display

The LED display lettering is shown in Figure 2-3 and has a length of 24 characters. The display is covered with a red filter printed with the 2920 assembly field format (see Chapter 4).

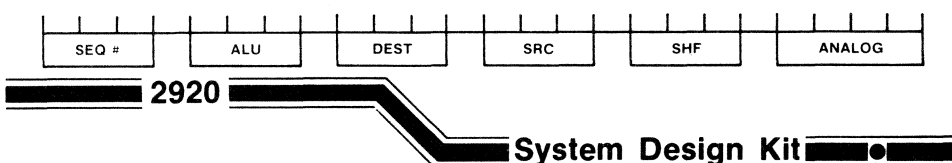


Figure 2-3. Display Format on LED Filter

0228

## Peripheral Interfaces

The peripheral sections are the serial port and audio cassette interface. Associated with the serial port are jumpers to select baud rate, jumpers to select line configurations and space for optional current loop. Chapter 5 describes the serial and cassette interfaces in detail.

## EPROM Programmer

The EPROM programmer is designed to read and program the 2920 EPROM. Circuitry associated with the EPROM programmer checks to verify correct socketing of the 2920 before applying voltages to help prevent damage to the device.



The 2920 should never be removed or inserted from the programming socket while the message "2920 TRANSFER ACTIVE" is displayed. Remove the 2920 from the programming socket when applying or removing power to the development side of the SDK-2920.



## Control Monitor

The monitor consists of an 8085A processor, ROM, RAM, and a host of support electronics. The monitor uses the key switches, display, peripheral interfaces, and EPROM programmer to perform the following functions:

- Edit and assemble 2920 programs
- Program the 2920 EPROM
- Transfer object files to and from audio cassette
- Transfer object files to and from development system
- Transfer source files to a development system or printer
- Convert numbers between binary and decimal

## Applications Side Physical Description

The applications section, shown in Figure 2-4, consists of a zero insertion force socket for a 2920 Signal Processor along with necessary support components. The board layout was designed to minimize cross talk and other noise. The 74S04 clock driver will accept one of two crystal inputs or a BNC connector input from an external frequency source. The board is laid out to accept up to four input and four output 2912 digital filters, each with their own voltage controlled oscillator (VCO) clocks. Two 2912's and VCO's are supplied with the kit.

Additional areas are provided for user designed input and output circuitry.

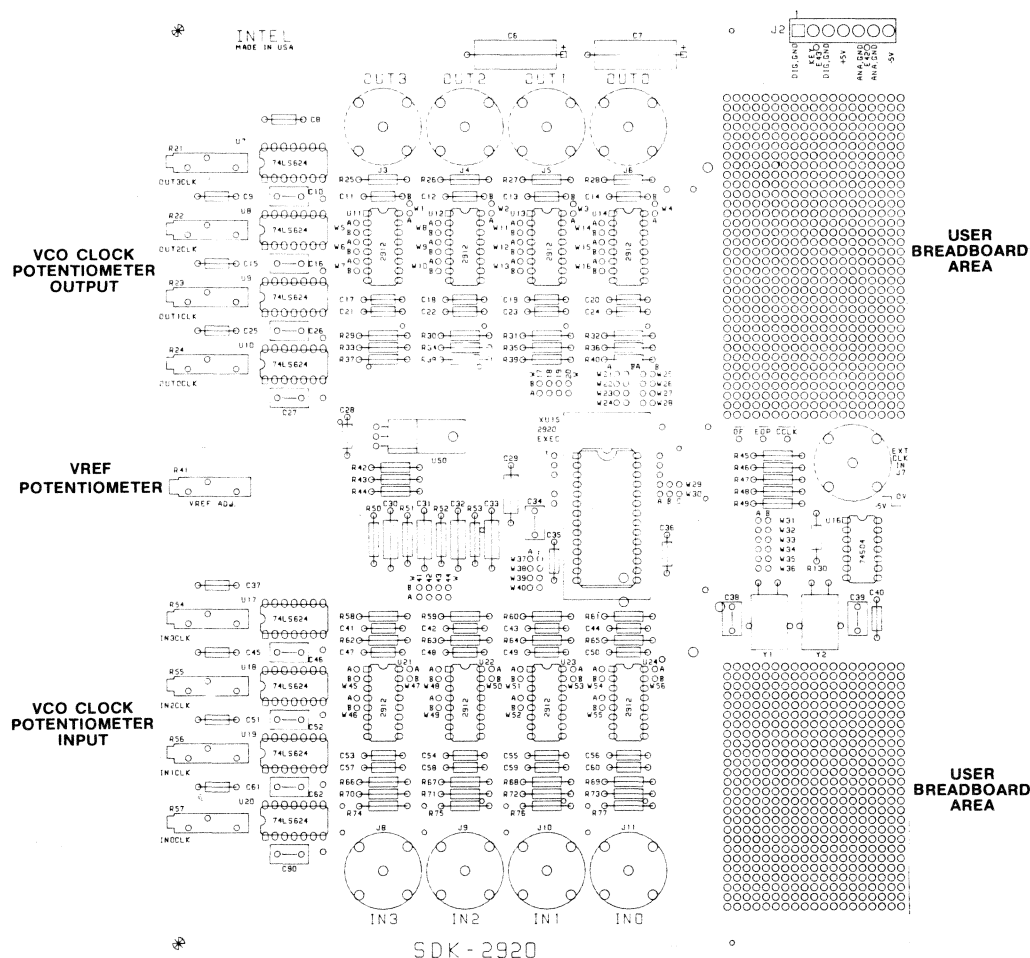


Figure 2-4. Applications Side

0229



## Introduction

The monitor is designed to prompt the user in all areas except EDIT. For example, the initial display "EDIT LOAD LIST CONV" indicates that only one of the four keys EDIT, LOAD, LIST or CONV can be pressed. After detecting a valid key closure the next prompt is displayed showing what options are legal. When EDIT is invoked, the prompt format is changed to a display oriented edit mode. The basic monitor structure is shown in Figure 3-1.

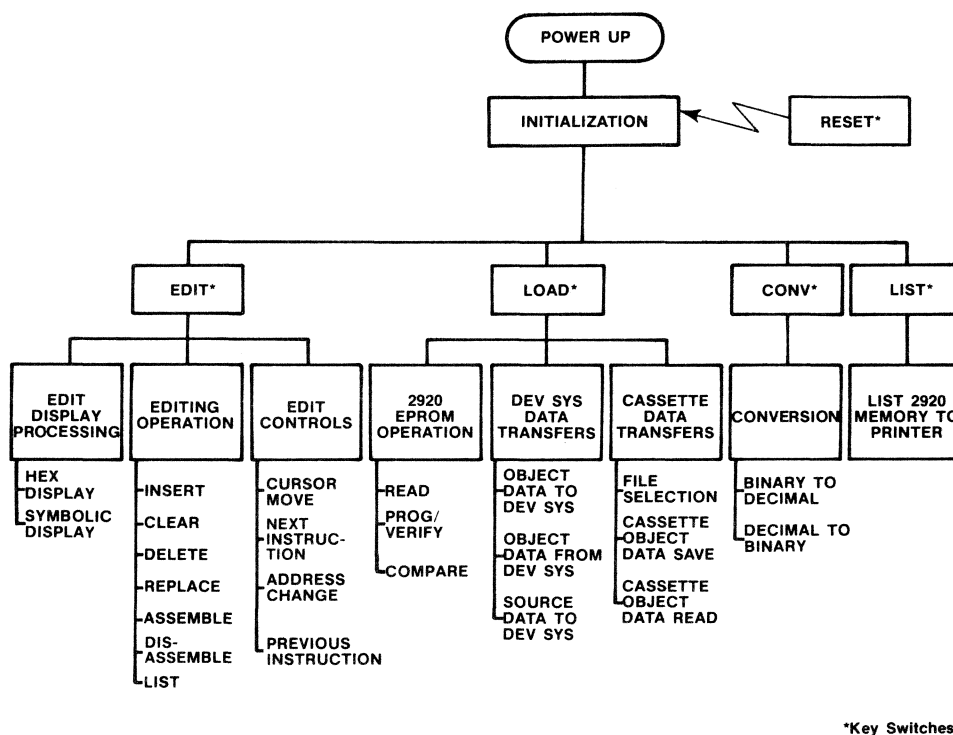


Figure 3-1. Monitor Structure

0230

Most monitor commands interact with the instruction memory (Figure 3-2). The instruction memory is 976 bytes of RAM within the SDK-2920 Design Kit. At power up the instruction memory is filled with a sample 2920 program which remains until cleared. Any loading of data into the SDK-2920 clears the sample program. When instruction memory is cleared it contains 192 occurrences of "4000EF" in hex or "LDA Y00 Y00 R00 NOP" in symbolic format. Instruction memory can contain only one program at a time.

Commands which modify instruction memory are:

- Edit, clear and modify
- Read 2920 EPROM
- Read object data from development system
- Read object data from cassette

Commands which access data in instruction memory are:

- Program/Verify 2920 EPROM
- Compare 2920 EPROM
- Transfer object data to development system
- Transfer source data to development system
- Transfer object data to cassette
- Edit, display option
- List to printer

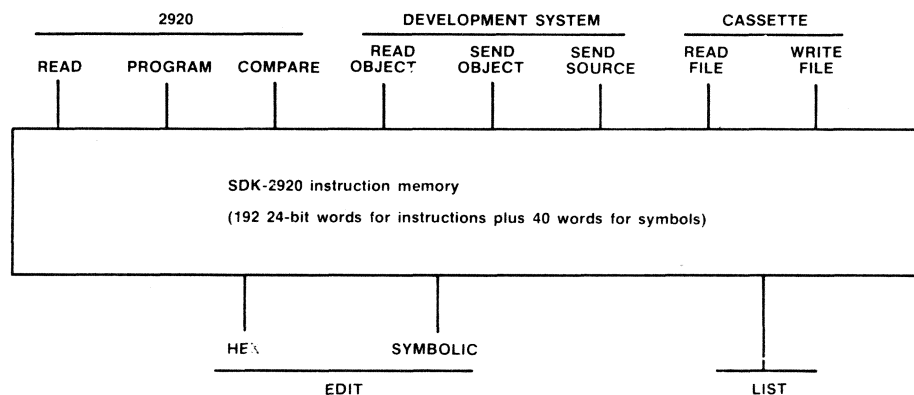


Figure 3-2. Command Interactions with Instruction Memory

0231

## Monitor Power Up

At power up, the monitor moves a sample program to instruction memory then performs a RESET (see next section).



Remove the 2920 from the programming socket before applying or removing power to the development side.

## Reset

### Function

The RESET command is used to terminate any operation in process. It is both a normal exit and error exit from any monitor operation.

### Invoking Command

**RESET** The RESET command is a single key which can be pressed at any time.

### Operation

The dedicated key RESET can be used to initiate a reset at any time. Reset is the highest priority operation and cannot be prevented by the monitor. Pressing the RESET key will interrupt any operation in progress and performs the following functions:

- Remove all power to the 2920 programming socket.
- Initialize the cassette and RS-232 ports.
- Display the monitor title and version number.
- Set the baud rate for RS-232 port as selected.
- Display the message "EDIT LOAD LIST CONV" and wait for a user key closure.

### Example

Key	Display
<b>RESET</b>	SDK-2920 Monitor VER x.y EDIT LOAD LIST CONV

## Editing Programs

The editor combines both the edit and assemble functions. The assembler executes after each 2920 instruction is entered but is not visible unless an error occurs. Errors detected by the assembler are displayed after each line of text is entered.

Pressing the EDIT key will invoke the editor any time the message "EDIT LOAD LIST CONV" is displayed. Once invoked, the edit allows the following operations:

- Fill instruction memory with NOP's and clear symbol table (clear command)
- Display and modify symbolic 2920 instructions (symbolic edit or modify command)
- Display and modify 2920 instructions displayed in hexadecimal (hex edit)

### Clear Command

#### Function

The internal storage of 2920 instruction is cleared to allow entry of a new 2920 program.

### Invoking Command

Key Pressed    Display

<b>RESET</b>	EDIT LOAD LIST CONV
<b>EDIT</b>	MODIFY = 1    CLEAR = 2
<b>2</b>	000 LDA Y00 Y00 R00 NOP

### Operation

The clear command fills 2920 instruction memory with “LDA Y00 Y00 R00 NOP” and clears the symbol table. Loading data into the memory from any peripheral device will automatically execute a clear operation first.

## Symbolic Edit (Modify Command)

### Function

The symbolic editor allows you to enter 2920 programs and make changes to existing programs.

### Invoking Command

Key Pressed    Display

<b>RESET</b>	EDIT LOAD LIST CONV
<b>EDIT</b>	MODIFY = 1    CLEAR = 2
<b>1</b>	

### Operation

To modify the currently displayed 2920 instruction, press one of the data keys on the right. The display indicates the data insertion point with a blinking character (cursor position). After the desired data has been entered, the following keys will cause the assembler to execute.

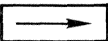
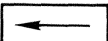



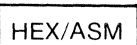


<b>PREV</b>	
<b>NEXT</b>	
<b>INSRT</b>	
<b>HEX/ASM</b>	(if in symbolic mode)

(Any change to SEQ or address field)

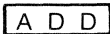
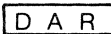
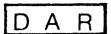

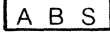


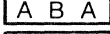


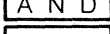

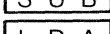

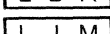

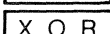



## NOTE

If the assembler detects an error, it does not change the operation requested. For example, if NEXT was pressed, and then an assembler error was detected, the display continues to the next instruction. The instruction to which the error applies is not changed. In order to make the intended change, you must press the PREV key to go back to that instruction.

The control keys used by EDIT are:

Key	Name	Description
	Cursor Right	The blinking cursor is moved right one position unless already at the right edge of displayed field.
	Cursor Left	Blinking cursor is moved left one character until the sequence number is encountered, then it skips to the left edge of display.
	Next instruction	The next 2920 instruction is displayed unless at end of memory.
	Previous Instruction	The previous 2920 instruction is displayed unless at beginning of memory.
	List Memory	Send disassembled 2920 instructions to serial port, followed by symbol table and hex dump of memory.
	Mode Toggle	Toggle edit mode between symbolic assembly and hexadecimal format.
	Insert Instruction	Expand the program in memory by one location and insert a NOP at current memory display address.
	Delete Instruction	Contract the program in memory by one location and remove the instruction at the current memory display position.

The right half of the keyboard contains data keys which can be entered at the blinking cursor position. Chapter 4 describes the 2920 fields and symbols in more detail. Recommended entries in each field are shown in figure 3-3.

SEQ	ALU	DEST	SRC	S H F	ANALOG
0 n n				R 0 0	
1 n n		Y Y Y	 n	•	
		A A A	 n	R 1 3	
		B B B	Y Y Y	L 0 1	
		C C C	A A A	L 0 2	 n
		D D D	B B B		 n
		E E E	C C C		 n
		F F F	D D D		 n
		n n	E E E		
			F F F		
			n n		

NOTE: "n" is a decimal digit. See Chapter 6 for limitations.

Figure 3-3. Instruction Fields and Legal Entries

0232

The symbolic edit feature allows you to enter a program in symbolic format or symbolically disassemble binary 2920 instructions. In either case the display will appear as shown in figure 3-4.

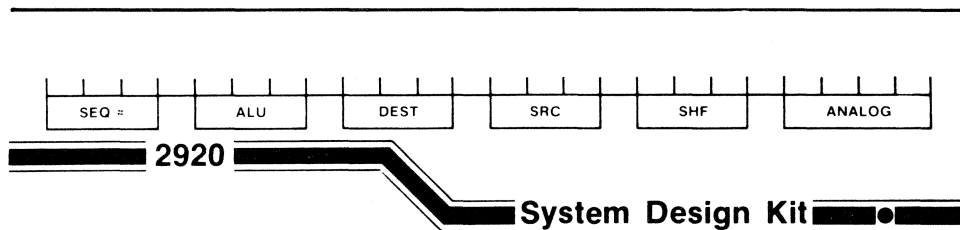


Figure 3-4. Display Panel

0233

One or more positions on the display will blink, indicating the cursor position. You can change the data at the cursor by pressing a legal data key. The entry of data plus moving to another instruction causes the assembler to execute. The assembler enters the new data into instruction memory if no errors were detected. Errors are displayed for a short period so you need to watch the display during data entry.

## NOTE

If any error is detected, the entire instruction is left unchanged. If the NEXT command is used to invoke the assembler, and an error results, the NEXT command is still executed, so that the instruction in the display is the next instruction rather than the one that resulted in the error.

## Example

To enter one instruction containing an EOP at location 005 (assumes RESET was pressed previously; underline indicates cursor position):

Key	Display	Comment
<b>EDIT</b>	MODIFY = 1    CLEAR = 2	
<b>1</b>	000 <u>SUB</u> Y00 KP1 L01 NOP-	modify memory contents
<b>←</b>	000 SUB Y00 KP1 L01 NOP	cursor left
<b>005</b>	005 <u>SUB</u> Y01 KP4 R00 NOP-	change sequence
<b>→</b>	005 SUB Y01 KP4 R00 NOP-	move cursor right
<b>⋮</b>		
<b>→</b>	005 SUB Y01 KP4 R00 <u>NOP-</u>	
<b>EOP</b>	005 SUB Y01 KP4 R00 <u>EOP-</u>	enter EOP
<b>NEXT</b>	006 <u>ABS</u> Y01 Y01 L01 CVTS	causes last data entry to be assembled
<b>RESET</b>	EDIT LOAD LIST CONV	exit EDIT



Another method of modifying location 005 to EOP:

Key	Display	Comment
<b>EDIT</b>	MODIFY = 1    CLEAR = 2	
<b>1</b>	000 <u>SUB</u> Y00 KP1 L01 NOP	
<b>NEXT</b>	001	Move one sequence number at a time to SEQ 005.
<b>NEXT</b>	002	
⋮		
<b>NEXT</b>	005 <u>SUB</u> Y01 KP4 R00 NOP	
<b>→</b>	005 SUB <u>Y</u> 01 KP4 R00 NOP	
⋮		
<b>→</b>	005 SUB Y01 KP4 R00 <u>NOP</u>	
<b>EOP</b>	005 SUB Y01 KP4 R00 <u>EOP</u>	
<b>NEXT</b>	006 <u>ABS</u> Y01 Y01 L01 CVTS	Executes assembler.
<b>RESET</b>	EDIT LOAD LIST CONV	

Insert an instruction at location zero, then delete it.

Key	Display	Comment
<b>EDIT</b>	MODIFY =    CLEAR = 2	
<b>1</b>	000 <u>SUB</u> Y00 KP1 L01 NOP-	
<b>INSRT</b>	000 <u>LDA</u> Y00 Y00 R00 NOP-	Inserted NOP.
<b>DEL</b>	000 <u>SUB</u> Y00 KP1 L01 NOP	

NOTE: Cursor position is indicated by underline.

## Hex Edit

### Function

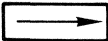
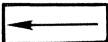



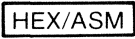


Hex edit allows entry of 2920 instructions in hexadecimal format. Entered instructions are also displayed in Hex format.

### Invoking Command

Key	Display	Comment
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>EDIT</b>	MODIFY = 1    CLEAR = 2	
<b>1</b> or <b>2</b>	(symbolic instruction)	Symbolic edit mode.
<b>HEX/ASM</b>	000 XXXXXX	

## Operation

The following control keys are used with Hex Edit.

Key	Name	Description
	Cursor Right	The blinking cursor is moved right one position unless at the end of displayed field.
	Cursor Left	Blinking cursor is moved left until the sequence number is encountered, then it skips to the left edge of the display.
	Next Instruction	The next 2920 instruction is displayed unless at end of memory.
	Previous Instruction	The previous 2920 instruction is displayed unless at beginning of memory.
	List Memory	Send disassembled 2920 instructions to serial port, followed by symbol table and hex dump of memory.
	Mode Toggle	Toggle edit mode between symbolic assembly and hexadecimal.
	Insert Instruction	Expand the program in memory by one location and insert a NOP at current memory display address.
	Delete Instruction	Contract the program in memory by one location and remove the instruction at the current memory display position.

The display format and legal data entry key is as follows:

Address	
SEQ	Hexadecimal
0 n n	XXXXXX
1 n n	

where "n" is a decimal number zero through nine

"X" is a hexadecimal number zero through F

The control keys can be pressed at any time and perform the same functions in symbolic edit and hex edit. The data keys are entered at the current cursor position. If a 2920 program is entered using the hex edit mode, then a symbol table is not generated until a disassembly is performed by one of the following: LIST, Symbolic Edit, Transfer source to development system, or LOAD object file to cassette. Before editing programs entered in hex, both symbol table operation and assembly formats should be studied (see Chapter 4).

## NOTE

Error checking in HEX mode is limited to the symbol table; thus errors are not flagged until the symbol table is generated. Since the hex data is not assembled, it can contain errors that are not detected by the hex edit. One way to check for such errors is to switch to the assembly mode (press HEX/ASM), then back to HEX mode. If errors are present, a three-second error message is displayed.

**Example**

Clear memory and enter one hex instruction at location 001 (underline indicates cursor position).

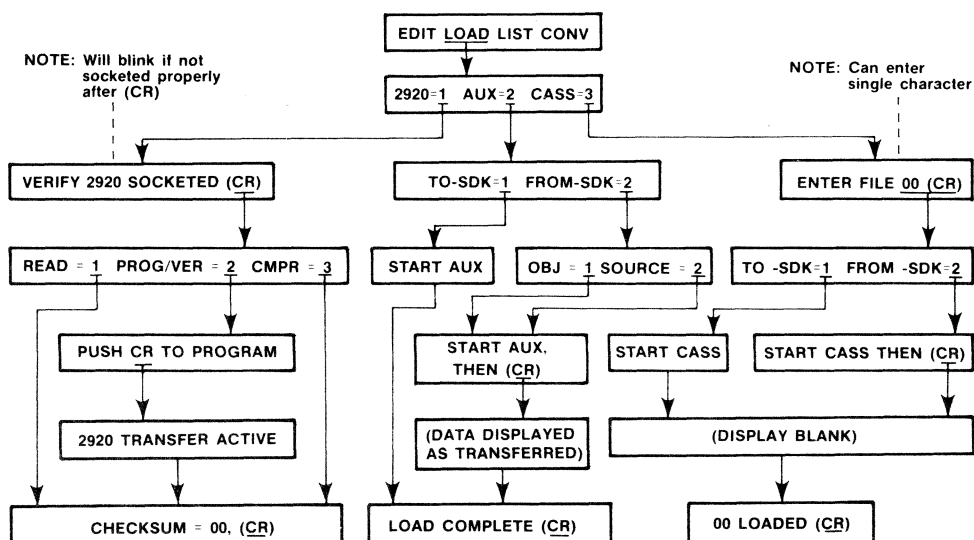
Key	Display	Comment
<b>EDIT</b>	MODIFY = 1    CLEAR = 2	
<b>2</b>	000 <u>LDA</u> Y00 Y00 R00 NOP	2920 NOP instruction
<b>HEX/ASM</b>	000 <u>4000EF</u>	2920 NOP instruction
<b>NEXT</b>	001 <u>4000EF</u>	
40223E	001 40223 <u>E</u>	location 1 now contains 40223E

Clear memory and enter a three instruction program.

Key Pressed	Display	Comment
<b>EDIT</b>	MODIFY = 1    CLEAR = 2	Edit mode entered
<b>2</b>	000 <u>LDA</u> Y00 Y00 R00 NOP	Clear memory
<b>HEX/ASM</b>	000 <u>4000EF</u>	Select hex edit mode
4008EF	000 4008 <u>EF</u>	new hex contents of loc 000
<b>NEXT</b>	001 <u>4000EF</u>	
40223E	001 40223 <u>E</u>	new hex contents of loc 001
<b>NEXT</b>	002 <u>4000EF</u>	
5008EF	002 5008 <u>EF</u>	new hex contents of loc 002

**Loading Programs**

The term is defined as any data transfer into or out of instruction memory and is invoked by pressing the LOAD key. One additional load operation is possible and can be invoked by pressing the LIST key. This section will describe the operation of both keys. The LOAD key will display continuous prompting messages until an operation is complete. Figure 3-5 shows the LOAD tree including all normal displays and keys pressed. The normal sequence for a successful 2920 EPROM read is shown by following the arrows on the leftmost portion of Figure 3-5.



**Figure 3-5. LOAD Command and Display Tree**

0234

## List To Printer

### Function

After entering a program into the 2920 instruction memory, the LIST command can be used to obtain a hard copy or CRT display. LIST command output data is sent to the serial port at jumper selected baud rate.

### Invoking Command

LIST can be activated within EDIT or following a RESET. Prior to pressing the LIST key all connections should be made to the serial port and the correct baud rate selected (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LIST</b>	(data is displayed as output)
	EDIT LOAD LIST CONV
<b>EDIT</b>	MODIFY = 1    CLEAR = 2
<b>1</b>	000 LDA Y00 Y00 R00 NOP
<b>LIST</b>	(data is displayed as output)
	EDIT LOAD LIST CONV

### Operation

The LIST command causes the following data to be output to both the display and the serial port:

- 1) 2920 instruction disassembly
- 2) Symbol table contents listed in sequential address assigned to each symbol.  
The format for each symbol is decimal address followed by symbol.
- 3) Hexadecimal dump of memory.

After all data has been output to the serial port, LIST returns to the RESET condition. Pressing the LIST command at 110 baud rate with or without a printer attached provides a sequential display of 2920 instruction memory.

### Example

List a three line program to display and printer. (Assumes program has been entered previously.)

Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	Disassembly of program plus three instructions past the EOP.
<b>LIST</b>	000 LDA Y00 Y01 R00 NOP	
	001 LDA Y00 Y01 R00 NOP	
	002 LDA Y00 Y01 R00 EOP	
	003 LDA Y00 Y01 R00 NOP	
	004 LDA Y00 Y01 R00 NOP	
	005 LDA Y00 Y01 R00 NOP	

Key	Display	Comments
	000 Y00	}
	001 Y01	
	040 DAR	
	4008EF	}
	4008EF	
	5008EF	
	4008EF	
	4008EF	
	4008EF	

## Read 2920 EPROM

### Function

Programs stored on the 2920 EPROM can be read into the SDK-2920 instruction memory by this command. This allows programs to be read, then modified and written back to the 2920 EPROM

### Invoking Command

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>1</b>	VERIFY 2920 SOCKETED <CR>
<b>CR</b>	READ = 1    PROG/VER = 2    CMPR =3
<b>1</b>	XXX = CHECKSUM <CR>

## NOTE

If the message "VERIFY 2920 SOCKETED" is blinking, then the 2920 is reversed in its socket.

### Operation

The 2920 instruction memory is cleared, then EPROM is read until the end of memory (192 instructions) or an EOP is encountered. EOP terminations always read three instructions past the EOP. If the EPROM is erased, the message "EPROM BLANK" is displayed.



Do not insert or remove the 2920 while its EPROM is being accessed. However, do remove the 2920 when applying or removing power to the development side.

### Example

Read 2920 EPROM into memory then enter EDIT mode.

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1   AUX = 2   CASS = 3
<b>1</b>	VERIFY 2920 SOCKETED (CR)
<b>CR</b>	READ = 1   PROG/VER =2   CMPR =3
<b>1</b>	XXX = CHECKSUM (CR)
<b>RESET</b>	EDIT LOAD LIST CONV
<b>EDIT</b>	MODIFY = 1   CLEAR = 2
<b>1</b>	000 SUB Y00, KP1, L01, NOP

## Program/Verify 2920 EPROM

### Function

Programs the 2920 EPROM with the data in instruction memory. As each byte is written to the EPROM, it is read back and verified.



Remove the 2920 from the programming socket when applying or removing power to the development side.

### Invoking Command

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1   AUX = 2   CASS = 3
<b>1</b>	VERIFY 2920 SOCKETED <CR>
<b>CR</b>	READ =1   PROG/VER = 2   CMPR =3
<b>2</b>	PRESS CR TO PROGRAM
<b>CR</b>	2920 TRANSFER ACTIVE

## NOTE

If 2920 is reverse socketed then the message 'VERIFY 2920 SOCKETED = (CR)' begins blinking and is held on display.

### Operation

First, the SDK-2920 instruction memory is checked to verify the program does not contain common errors. Appendix C describes error messages. After any error, press **CR** to have the system continue checking for errors, or press **0** to bypass further error checking.

Next, the 2920 EPROM is checked to verify that it is erased or has bits programmed in the same position as the program in the SDK-2920 Design Kit instruction memory. If this check fails the message "STUCK BIT XXX" is displayed, indicating the location of the error. If the stuck bit test passes and the program contains an EOP, a check is made to verify the 25V supply is working.

If the 25V test passes the programming operation is initiated. After each write operation to the 2920 EPROM, the data is read back and compared. If an error occurs then the message "ERR AT LOC XXX" is displayed. Successful compares are added to the checksum and if the programming operation is successful, then the message "XXX = CHECKSUM" is displayed.

**Example**

Program a 2920 EPROM then return to monitor.

Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3	
<b>1</b>	VERIFY 2920 SOCKETED <CR>	
<b>CR</b>	READ=1    PROG/VER=2    CMPR=3	
<b>2</b>	PRESS CR TO PROGRAM	if only code check operations desired, then press RESET
<b>CR</b>	2920 TRANSFER ACTIVE XXX=CHECKSUM <CR>	
<b>CR</b> or <b>RESET</b>	EDIT LOAD LIST CONV	

**Compare 2920 EPROM****Function**

The 2920 EPROM is compared to the SDK-2920 instruction memory.



Remove the 2920 from the programming socket when applying or removing power to the development side.

**Invoking Command**

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>1</b>	VERIFY 2920 SOCKETED <CR>
<b>CR</b>	READ=1    PROG/VER=2    CMPR=3
<b>3</b>	XXX = CHECKSUM <CR>

**Operation**

The 2920 EPROM is compared to the SDK-2920 instruction memory until either an EOP or an end of memory occurs. If an EOP occurs, then the compare continues for three more instructions. Miscompares result in the message "ERR AT LOC XXX", otherwise the checksum is displayed.

**Example**

Compare EPROM to SDK-2920 instruction memory then exit.

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>1</b>	VERIFY 2920 SOCKETED <CR>
<b>CR</b>	READ = 1    PROG/VER = 2    CMPR = 3
<b>3</b>	XXX = CHECKSUM <CR>
<b>CR</b> or <b>RESET</b>	EDIT LOAD LIST CONV

**Read Object Data from Development System**

*TO SDK*

**Function**

Reads object files created by the resident Intellec assembler or SDK-2920 Design Kit. The data read is stored in SDK-2920 instruction memory.

**Invoking Command**

Verify the development system is connected correctly (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>1</b>	START AUX

**Operation**

The read routine utilizes the hex object code output format (see Chapter 5). Initially it waits for a block start, “:”, and continues to read and add data to the checksum until an end block is detected. The message START AUX stays on the display during the load. If no errors are encountered, the message LOAD COMPLETE is displayed. Possible problems are:

- 1) Data not received (no error message displayed) — check baud straps, cable, SDK-2920 straps.
- 2) Checksum error displayed.
- 3) Incorrect data format (usually results in a checksum error).

**NOTE**

Before downloading any program it should be free of assembly errors; otherwise it should be disassembled and checked for errors.



**Example**

Read one file from development system:

- Connect cable, set baud, set jumpers (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>1</b>	START AUX

- Type the following on the development system  
COPY :FX:file name TO :TO:(CR)  
(Where :FX: is the drive number where the file resides.)
- SDK-2920 will display load complete when done.

**Transfer Object Data To Development System**

= FROM SDK

**Function**

Converts data in the SDK-2920 instruction memory into object format and transfers it to a development system via the serial port.

**Invoking Command**

Verify the Intellec is connected correctly (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>2</b>	OBJ = 1    SOURCE = 2
<b>1</b>	START AUX, THEN (CR)

**Operation**

The transfer program utilizes the hex object code output format (see Chapter 5). Since no handshaking occurs, the data is sent even if the serial port is not connected. During the data transfer, the display will flash with data until the "LOAD COMPLETE" message appears.

**Example**

Connect cable, set baud, set serial port straps, then enter the following:

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>2</b>	OBJ = 1    SOURCE = 2
<b>1</b>	START AUX, THEN (CR)

Enter on the development system console "COPY :TI: TO:FX:filename (CR)" (Where :FX: filename is the device and file to receive the data.)

<b>CR</b>	(data appears on display)
	LOAD COMPLETE

**Transfer Source Data to Development System**

← FROM SDK

**Function**

The SDK-2920 instruction memory is disassembled and sent to the serial port. This command allows 2920 programs to be sent to the Intellec for EDIT's and assemblies.

**Invoking Command**

Verify the development system is connected (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>2</b>	OBJ = 1    SOURCE = 2
<b>2</b>	START AUX, THEN (CR)

**Operation**

The SDK-2920 instruction memory is disassembled and sent to the serial port and to the display. At the end of the transfer the system sends a control Z character to the Intellec to terminate the transfer.

**Example**

Connect cable, set baud, set serial port straps as described in Section 5.1

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>2</b>	TO-SDK = 1    FROM-SDK = 2
<b>2</b>	OBJ = 1    SOURCE = 2
<b>2</b>	START AUX THEN (CR)

Enter on development system console "COPY :TI: TO :FX: filename (CR)"

**CR** (display will show data as it is sent)  
LOAD COMPLETE  
**CR** EDIT LOAD LIST CONV

## Read Object Data From Cassette

### Function

One object file on cassette is read into the SDK-2920 instruction memory. Also, the symbol table is read along with the file.

### Invoking Command

Verify the cassette is connected (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3
<b>3</b>	ENTER FILE 00(CR)
<b>1</b>	ENTER FILE 01(CR)
<b>CR</b>	TO-SDK = 1    FROM-SDK = 2
<b>1</b>	START CASS
	(display will blank and remain blank until transfer is complete)

### Operation

Reading the cassette requires manually starting and stopping the tape (in play or forward mode) plus the cassette must contain hex object files which match the file name entered. Legal file names can be any combination of two decimal numbers. The tape is read until a file name match is found. The data after the file name is then stored into the instruction memory and symbol table. Successful reads are followed with the message "XX LOADED". If errors occur, the message "CHECK-SUM ERROR" is displayed. Usually errors can be overcome by adjusting cassette volume or checking electrical hookup and trying the operation again. (On some machines it may not be permissible to have the microphone and monitor jacks in place at the same time.)

## NOTE

When an incorrect file name is detected it is displayed for two seconds before continuing the search for the file requested.



Before reading a cassette file, adjust the cassette volume by starting the cassette, watching the red LED near the cassette connector, and positioning the volume control so that the LED just begins to flicker. If the signal level from the recorder is too high, noise errors will be introduced into the system.

### Example

Read file 75 from cassette then exit. See Chapter 5 for description of cassette connections.

Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>LOAD</b>	2920 = 1 AUX = 2 CASS = 3	
<b>3</b>	ENTER FILE 00(CR)	
<b>7</b>	ENTER FILE 07(CR)	If desired, the file number can be
<b>5</b>	ENTER FILE 75(CR)	changed or connected before press
<b>CR</b>	TO-SDK = 1 FROM-SDK = 2 ing (CR).	
<b>1</b>	START CASS	
	(display will blank after three seconds; wait until it goes blank before starting the cassette.)	
	(start cassette in play mode)	
	75 LOADED	
<b>RESET</b>	EDIT LOAD LIST CONV	

## Transmit Object Data To Cassette

### Function

One object file plus symbol table is moved from the SDK-2920 instruction memory to the cassette.

### Invoking Command

Verify cassette is connected (see Chapter 5).

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920 = 1 AUX = 2 CASS = 3
<b>3</b>	ENTER FILE 00(CR)
<b>1</b>	ENTER FILE 01(CR)
<b>CR</b>	TO-SDK = 1 FROM-SDK = 2
<b>2</b>	START CASS, THEN (CR)

### Operation

The cassette tape must be manually positioned and started in the record mode. Then the monitor outputs the instruction memory and symbol table to cassette when CR is pressed. Each block of data sent to the cassette contains a check byte which is used to verify correct data reception when the data is read back.

### CAUTION

Be sure the cassette is positioned past the leader before attempting to write. On some cassettes the earphone and microphone plugs should not be plugged in at the same time, because of increased noise levels induced in the data transfer.

### Example

Write file 75 to cassette.

- 1) Connect the cassette as described in Chapter 5.
- 2) Write the file in program memory to the cassette, and stop the cassette when message "XX LOADED <CR>" occurs.

3) Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>LOAD</b>	2920 = 1    AUX = 2    CASS = 3	
<b>3</b>	ENTER FILE 00(CR)	CR can be entered after one character has been entered.
<b>7</b>	ENTER FILE 07(CR)	
<b>5</b>	ENTER FILE 75(CR)	
<b>CR</b>	TO-SDK = 1    FROM-SDK = 2	
<b>2</b>	START CASS, THEN (CR)	

- 4) Start cassette in record mode. (Allow for leader if at the beginning of the tape).

5) Key	Display
<b>CR</b>	(display is blank during load) 75 LOADED <CR>
<b>RESET</b>	EDIT LOAD LIST CONV

- 6) Stop cassette.

## Converting Number Bases

The convert key is provided to assist with generating constants within the 2920 program. The program will be initially conceived using decimal values then converted to binary and built in the 2920 using a combination of shifts, adds and subtracts. The convert routine calculates decimal numbers in the range  $\pm 4194303$ .

### Convert Binary to Decimal

#### Function

Converts signed integers or fractional binary numbers to decimal.

#### Invoking Command

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>CONV</b>	ENTER NUMBER
<b>B</b>	B

## Operation

The monitor expects the first entry following "B" to be a sign bit. For negative numbers the sign bit is "1". Following the sign, up to twenty-two "1's" or "0's" can be entered. When the proper number of binary characters have been entered, a **CR** initiates the conversion. If a binary point is entered, the maximum number size is reduced to 22 bits. Positive integers over 18 bits require up to one minute to perform the conversion. Following the conversion, a new number may be entered by pressing the "B" or "D" keys. Errors cause the message "ENTER NUMBER" to be displayed again. The only limit on the size of binary numbers to be converted is the display length.

## Example

Convert 0101.101 to decimal, then check to verify binary was entered correctly.

Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>CONV</b>	ENTER NUMBER	Convert prompt
<b>B</b>	B	Binary to decimal
	0101.101	
<b>CR</b>	D+5.62500000	Converted number
<b>CR</b>	B0101.101000000000000000	Convert back to original entry.

A new binary or decimal number can be entered by pressing the "B" or "D" keys.

## Convert Decimal to Binary

### Function

Converts signed integers or fractional decimal numbers to binary.

### Invoking Command

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>CONV</b>	ENTER NUMBER
<b>D</b>	D+

The monitor expects the first entry following the "D" to be a sign change, decimal point or decimal digit. The maximum and minimum values which can be entered are shown in Table 3-1. After the number has been entered a "CR" is required to start the conversion. If too large a value is entered, the conversion process is aborted and the message "ENTER NUMBER" is displayed.

**Table 3-1. Conversion Limits**

Smallest positive number +.00000023
Largest positive number +4194303
Smallest negative number -.00000047
Largest negative number -4194304

**Example**

Convert 1024.125 to binary then back to decimal.

Key	Display	Comments
<b>RESET</b>	EDIT LOAD LIST CONV	
<b>CONV</b>	ENTER NUMBER	
<b>D</b>	D+	Conversion will be decimal to binary
1024.125	D+1024.125	
<b>CR</b>	B010000000000.0010000000	Converted result
<b>CR</b>	D+1024.12500000	Conversion from binary to decimal

A new binary or decimal number can be entered by pressing "B" or "D" following a conversion.







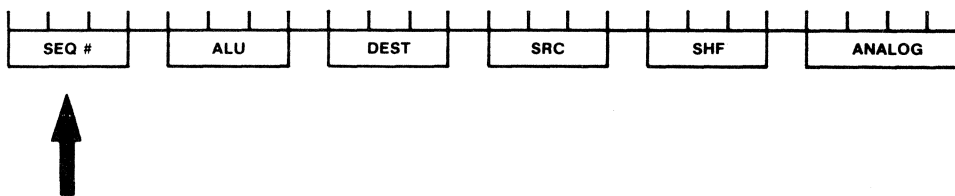
## How To Use This Chapter

This chapter lists the 2920 instructions and other information regarding entering instructions with the SDK-2920 design kit. The instructions are grouped according to the field in which they can appear (left-to-right order on the display).

### NOTE

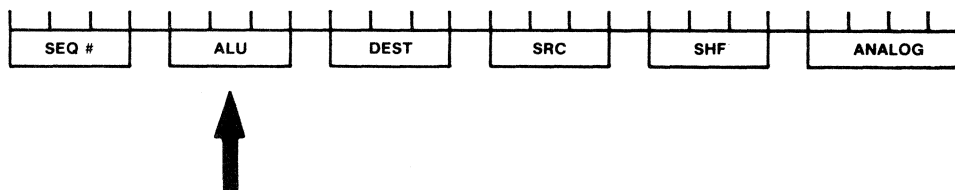
In the examples of instructions, the entry “AAA” refers to any valid entry in the source field (SRC or “A” field), and “BBB” represents any valid entry in the destination (DEST or “B” field).

## Sequence Field



The sequence field can contain any decimal value between 000 and 191. The decimal value shown indicates which 2920 instruction is displayed in the remaining fields. The display changes dynamically as the digits are entered left to right.

## ALU Field



The ALU field can contain one of the following single-key entries:

ABA	ADD	LDA	SUB
ABS	AND	LIM	XOR

### ABA — Absolute Value and Add

The absolute value of the source, after any shifting, is added to the destination and the result is stored in the destination.

The normal standard carry and overflow apply to ABA, as explained in Appendix E.

**Examples:**

ABA BBB,AAA,R00,NOP

This example takes the absolute value of the contents of AAA and adds that to the value in BBB, placing the result in BBB.

ABA BBB,AAA,L01,NOP

This example doubles the value from AAA by a left shift one position, then takes the absolute value of that result and adds it into BBB.

ABA BBB,AAA,L02,CND5

After shifting the value from AAA left two positions, effectively multiplying it by four, this command adds the absolute value of that result into BBB.

**ABS — Absolute Value**

This instruction takes the absolute value of the source operand, after any shifting, and stores it in the destination. If the source was positive, the destination becomes identical to the source. If it was negative, the destination is the “negative” of the source, that is, of same magnitude and opposite sign.

ABS never has a carry; it clears the carry flag to zero. A left-shift could cause overflow.

**Examples:**

ABS BBB,AAA,R00,NOP

This instruction places the absolute value of AAA into BBB. If AAA were 0.0000 0001, BBB would become 0.0000 0001. If AAA were 1.1111 1111, BBB would become 0.0000 0001.

ABS BBB,AAA,R01,NOP

This command shifts the value from AAA to the right 1 position, effectively halving that value, and then places the absolute value of this result into BBB.

**ADD — Addition**

After any shifting of the source operand, this instruction forms the sum of the source and the destination operands. The result is stored in the destination.

The normal standard carry and overflow apply, as explained in Appendix E.

**Examples:**

ADD BBB,AAA,R00,NOP

The sum of the contents of AAA and BBB will be placed in BBB.

ADD BBB,AAA,R01,NOP

The shiftcode R01 will shift the value from right one position, effectively halving it. This result will be added to the current contents of BBB.

**AND — Logical Conjunction**

After any shifting of the source operand, this instruction performs the logical AND of the shifted value with the value from the destination, and stores the result in the destination.

The normal standard carry and overflow apply, as explained in Appendix E.

**Examples:**

```
AND    BBB,AAA,R00,NOP
```

The value from AAA will be ANDed against the value from BBB, with the result of this logical operation placed into BBB.

```
AND    BBB,AAA,R01,NOP
```

After shifting the value from AAA to the right three positions, this instruction will AND the result with the value from BBB, storing the result of this logical operation back into BBB. The effect depends on the sign bit of AAA, since right shifting fills from the left with whatever the sign bit was, 1 if negative, 0 if positive.

**LDA — Load Source To Destination**

This instruction writes into the specified destination the value of the source operand after any shifting.

LDA always clears the carry. A left-shift could cause overflow.

**Examples:**

```
LDA    BBB,AAA,R01,NOP
```

This instruction writes into BBB half the value from AAA (because that value is right-shifted one bit position before the LDA gets it).

**LIM — Load Destination With Source Limit**

This instruction loads one of two extreme values into the destination, based on the sign of the source operand. If the source is positive or zero, the destination gets a plus 1 (0.11111111111111111111111111111111). If the source is negative, the destination gets a minus 1 (1.00000000000000000000000000000000).

The normal standard carry and overflow apply, as explained in Appendix E. LIM sets the carry to 0, and can have an overflow only via a left shift.

**Example:**

```
LIM    BBB,AAA,R00,NOP
```

The contents of AAA will be  $-1.0$  or  $+1.0$  depending on whether BBB is negative or not, respectively (zero being non-negative).

## SUB — Subtraction

This instruction subtracts the value in the source operand (after any shifting), from the value in the destination operand. Subtraction is done by adding the one's complement of the source and forcing a carry input at the lowest-order bit (see Appendix D).

The normal standard carry and overflow apply, as explained in Appendix E.

```
SUB    BBB,AAA,R00,NOP
```

Here the value from AAA is subtracted from the value in BBB, writing the result into BBB.

If a conditional analog field mnemonic is specified, the SUB instruction operates as either subtraction or addition. This special operation is discussed in the analog field section (CND instructions) later in this chapter.

## XOR — Exclusive OR Instruction

This instruction forms the exclusive OR of the source (after any shifting) with the destination, and stores the result in the destination.

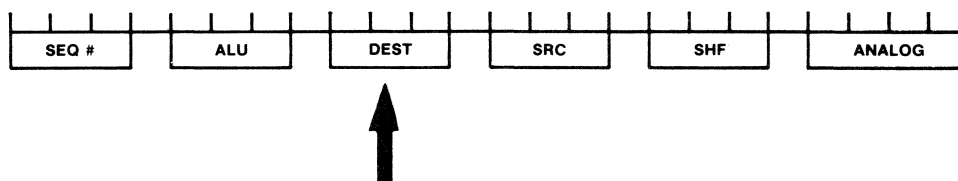
Exclusive OR gives a 1 in each bit position where only one of the two values has a 1, and gives a 0 in those bit positions where both have ones or both have zeroes.

```
XOR    BBB,AAA,R00,NOP
```

This will form the exclusive OR of the values in these two operands, and the result will be written into BBB.

XOR is implemented as an ADD with no carries. See Appendix E for further discussion of carry and overflow for XOR.

## Destination Field



The destination or B-field can contain either the single key entry DAR, or a three-character label that you define. Labels must begin with one of the characters A, B, C, D, E, F, or Y; the remaining characters may be any keyboard entry. Up to 40 labels may be defined, corresponding to the RAM locations. The system assigns locations to labels automatically; the address is only visible if the LIST command is invoked.

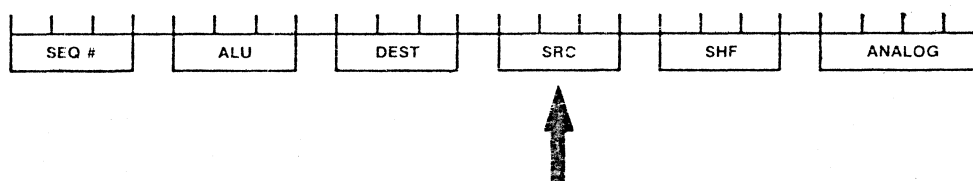
## NOTE

The symbol "Y00" is automatically placed in the symbol table when the editor is entered; thus only 39 labels can be defined by the user. If your program requires 40 unique labels, therefore, one of them must be "Y00".

## NOTE

Labels are placed in a symbol table anytime the assembler encounters a new label or the disassembler encounters a non-assigned field value. If the table overflows you should either delete the entire table and start again or re-use existing labels. The entire table can be deleted by reloading the program from 2920 EPROM, loading from a development system, or clearing and reentering the program. Alternatively, list the program to the display or printer, then select labels not currently used or used in another part of the program.

### Source Field

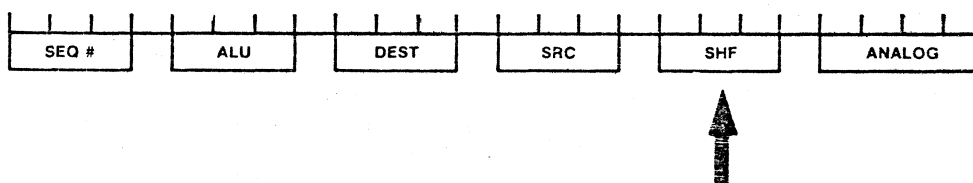


The source or A-field can contain the single-key entry DAR, a three-character label (as discussed previously under destination field), or one of the constants shown in Table 4-1.

Table 4-1. Constant Codes

CONSTANT MNEMONIC	UNSCALED VALUE	CONSTANT MNEMONIC	UNSCALED VALUE
KP0	0	KM1	- .125
KP1	+ .125	KM2	- .250
KP2	+ .250	KM3	- .375
KP3	+ .375	KM4	- .500
KP4	+ .500	KM5	- .625
KP5	+ .625	KM6	- .750
KP6	+ .750	KM7	- .875
KP7	+ .875	KM8	-1.0

### Shift Field



The shift field can contain one of the scaler codes shown in Table 4-2. Each of these shifts in effect multiplies the source field value by a power of two, where the number of positions shifted is the power. Shifting is completed before any arithmetic (ALU) operations are performed.

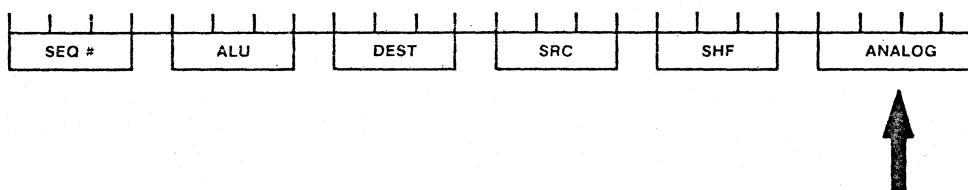
## NOTE

The system issues a warning when a shift code is used with a LIM instruction, but execution of the LIM instruction is not affected.

Table 4-2. Scaler Codes

SCALER CODE	EQUIVALENT MULTIPLIER	SCALER CODE	EQUIVALENT MULTIPLIER
L02	$2^2=4.0$	R06	$2^6=0.015625$
L01	$2^1=2.0$	R07	$2^7=0.0078125$
R00	$2^0=1.0$	R08	$2^8=0.00390625$
R01	$2^{-1}=0.5$	R09	$2^9=0.001953125$
R02	$2^{-2}=0.25$	R10	$2^{10}=0.0009765625$
R03	$2^{-3}=0.125$	R11	$2^{11}=0.00048828125$
R04	$2^{-4}=0.0625$	R12	$2^{12}=0.000244140625$
R05	$2^{-5}=0.03125$	R13	$2^{13}=0.0001220703125$

### Analog Field



The analog field can contain one of the following types of entries:

CNDn	EOP	NOP
CVTn	INn	OUTn

### CNDS, CND7, CND6, CND5, CND4, CND3, CND2, CND1, CND0

Each of these analog field entries refers to a single bit, either a bit of the DAR or (with SUB) the carry bit. CNDS means conditional on the sign bit; the others refer to specific bit positions in the DAR. CND0 refers to the least significant bit.

If the tested bit is a 1, the operation is performed as written. If the tested bit is a 0, the operation is either not performed at all or is altered. Only three ALU opcodes are affected: ADD, LDA, and SUB.

The normal standard carry and overflow apply, as explained in Appendix E. LDA, however, never has a carry.

### NOTE

The CND entries have no effect on the ABS, AND, or LIM operations; a warning is displayed when CND is used with one of these ALU operations. When a CND is used with the ABA instruction, the limiting effect of overflow detection is turned off; it turns on again after EOP or after an XOR with any CND entry (refer to Appendix E).

### ADD Conditional

ADD     BBB,AAA,R00,CNDS

If the sign bit of the DAR is 1, this instruction will add the contents of AAA to the contents of BBB and store the result in BBB.

If the sign bit of the DAR is 0, then the sum of BBB with zero is placed into BBB, i.e., no change except that the carry flag is cleared.

**LOAD Conditional**

LDA     BBB,AAA,R02,CND5

If bit five of the DAR is 1, this instruction will get the value of AAA, shift it right two positions to create  $\frac{1}{4}$  the value, and put it into BBB, writing over whatever value was formerly there.

If bit five is 0, the effect is the same as with the conditional add.

**SUBTRACT Conditional**

SUB     BBB,AAA,R00, CND0

Conditional subtract is a special operation, requiring information about the previous carry situation.

- If the carry resulting from the previous ALU operation is a 1, then the subtraction indicated is performed, i.e., the value from AAA is subtracted from the value in BBB, and the result is written into BBB.
- If the carry resulting from the prior ALU operation is 0, then the operands are added instead of being subtracted, i.e., the value from AAA is added to the value from BBB, and the sum is written into BBB.

The above instruction will set the first bit of the DAR, DAR(0), to the carry output of the highest order position of the ALU. Then, depending on the carry resulting from the previous ALU operation, it will perform either an addition or a subtraction.

**CVTS, CVT7, CVT6, CVT5, CVT4, CVT3, CVT2, CVT1, CVT0**

In order to convert to a digital value from an input sample value in the sample-and-hold for input, each of these analog field values sets the named bit of the DAR (e.g., bit 7 for CVT7) to 1 or 0 based on that input value. Each CVT also sets the next lower bit (e.g., bit 6 for CVT7) to 1 as part of the conversion process. The process uses the comparator and the reference voltage (VREF) to decide the sign and the fraction of VREF which represents the input sample. It is necessary to allow the DAC to settle between each cycle of conversion. This is achieved by inserting NOP analog field mnemonics after all CVT's, or placing CVT analog entries only on every other ALU instruction. Note: At some clock frequencies it may be necessary to include two NOP's between successive CVT instructions.

**EOP — End of Program**

EOP signals the end-of-program condition, causing a transfer back to the instruction in location zero. This analog field entry must be on in a location whose address is a multiple of four or a warning is issued. (If a program with a misplaced EOP is executed, the results are unpredictable.) The 2920 instruction words are pipelined in groups of four. The 2920 will execute three instructions after the EOP before branching back to location zero.

Overflow limiting is turned on by the execution of an EOP and thus is enabled during the last four instructions of the program.

**IN0, IN1, IN2, IN3 — Inputs**

You use these analog field entries to obtain an input sample from one of the four input channels. It is generally necessary to use a sequence of several INs in order to obtain a reliable sample. (The number of INs is a function of the sample capacitor and the clock rate.) For details on how many IN instructions are necessary, refer to the 2920 Analog Signal Processor Design Handbook.

**NOP — No-Operation**

NOP means no-operation for the analog section of the 2920 chip.





## CHAPTER 5 PERIPHERAL INTERFACES

### Introduction

Two interface ports are provided with the SDK-2920. One is a dedicated audio cassette interface and the other can be RS-232 or a current loop. In addition the serial RS-232 port can be configured as either a printer or Inteltec interface. See Figure 5-1.

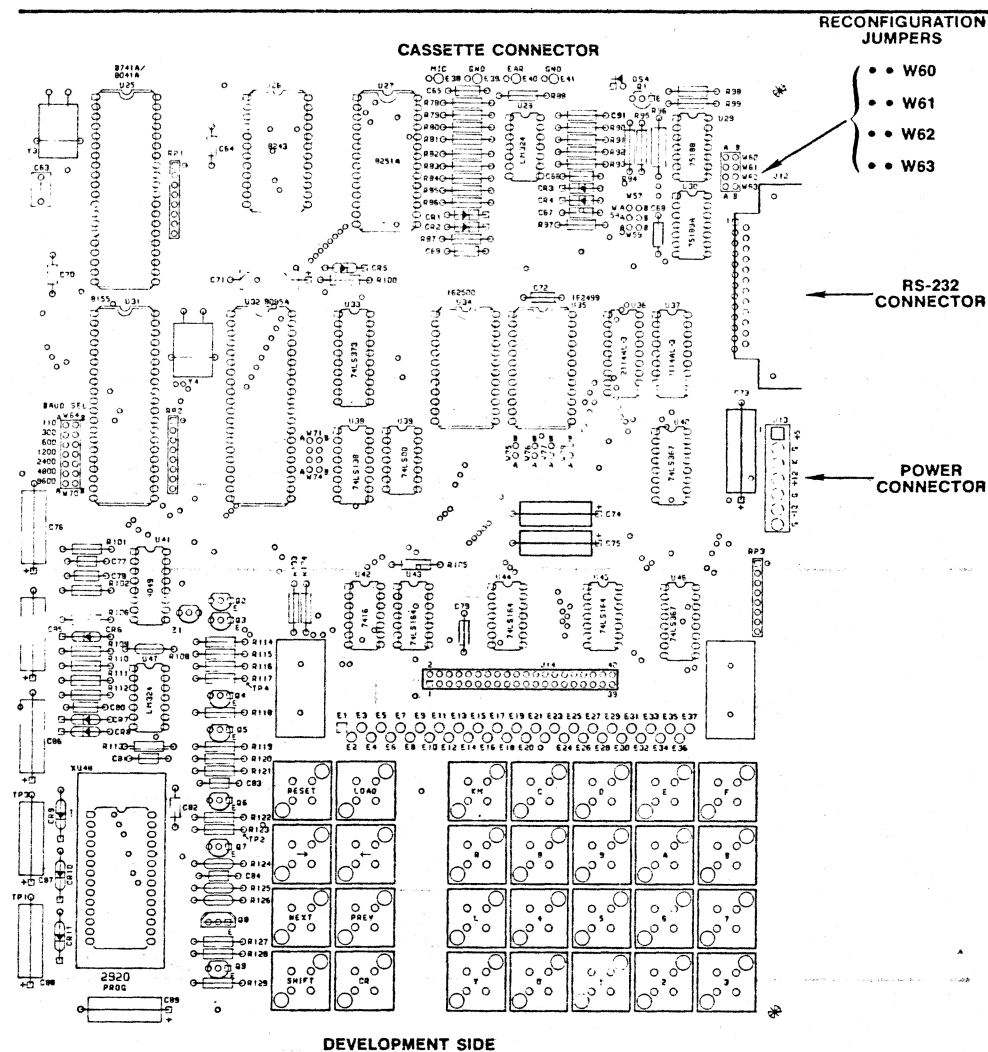


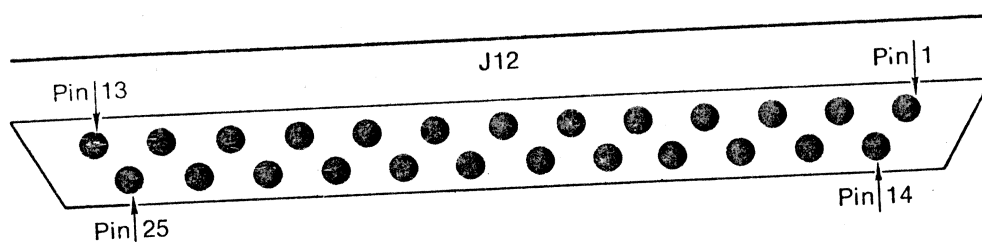
Figure 5-1. Peripheral Interface Location

0235

### Serial Transfer Port

The serial interface port can be configured for a development system, printer, or current loop device (Figure 5-2).

# Peripheral Interfaces



## RS-232

Pin 2 transmit data  
Pin 3 receive data  
Pin 4 request to send  
Pin 5 clear to send  
Pin 7 ground

## CURRENT LOOP

Pin 12 transmit data (+)  
Pin 13 receive data (+)  
Pin 24 transmit data (-)  
Pin 25 receive data (-)

Figure 5-2. RS-232 and Current Loop Pin Assignments

0236

The connector pin assignment can be changed by using the reconfiguration jumpers as described in the next two sections. The output/input rate is controlled by the baud select jumpers which can be set to the seven baud rates shown in figure 5-3.

rear  
of  
board



0 0	110
<b>0 0</b>	300
0 0	600
0 0	1200
0 0	2400
0 0	4800
0 0	9600

Figure 5-3. Baud Rate Selection Jumpers

0237

In the example above, 300 baud was selected by placing a jumper across the second row of connectors. After selecting the baud setting, always press the RESET key to initialize the baud change.

## Development System Interface Connections

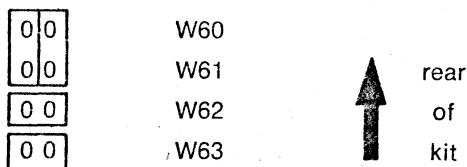
The following steps are needed to interface with a development system.

- 1) Set the baud rate jumper at 110 for Series II Development Systems. Set the baud rate jumper to agree with the setting of the baud rate jumper on the monitor module card in the 800 Series Development Systems.

- 2) Position the reconfiguration straps for RS-232 as follows:



If the receive and transmit pins are reversed on the RS-232, position as follows:



- 3) To interface to Series II:

- a) Using existing SDK serial port (RS-232) — connect to the serial 1 connector (J2) on the development system. Development system commands use :TO: , :TI:

*→ B1200  
→ 4,5 verbinden op RS232 conn. CTS/RTS*

To interface to 800 Series:

- a) Use conversion interface connected to the TTY port of the 800. Development system commands use :TI: , :TO:

OR:

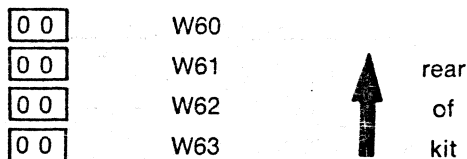
- b) Connect a teletype device to the TTY port, and disconnect your CRT from the video port. The video port is then free to interface directly to the RS-232 port on the kit. Development system commands use :VO: , :VI:

- 4) Press RESET key on SDK-2920 Design Kit.

## Printer Interface Connection

The following steps are needed to interface with a printer:

- 1) Set baud rate to match printer.  
2) Position the reconfiguration straps as follows:



If the printer requires RTS-CTS handshaking, remove the jumper from W62.

If the printer is a current loop device, remove the jumpers from W60, W61 and W63 and place them on W57, W58 and W59.

- 3) Connect standard cable to the printer and SDK-2920 Design Kit.  
4) Press RESET key on SDK-2920 Design Kit.

## Object Code Output Format

The object code generated by the development system assembler and the SDK-2920 is configured as follows:

- Field 0: Record mark (frame 0 is always ':')
- Field 1: Record length (frames 1 and 2)
- Field 2: Load address field (frames 3, 4, 5 and 6)
- Field 3: Record type (frames 7 and 8)
- Field 4: Data
- Field 5: Checksum (frames 'data field' +1 and 'data field' +2)

The format of the object code is a series of records, each containing its record length, type, memory load address, checksum, and data. The following figure shows a typical output file in hexadecimal format. The data field contains binary data when used with cassettes, and ASCII hexadecimal when used with a development system.

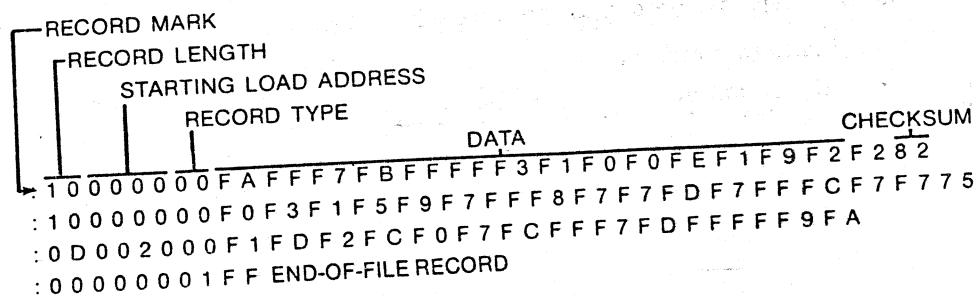
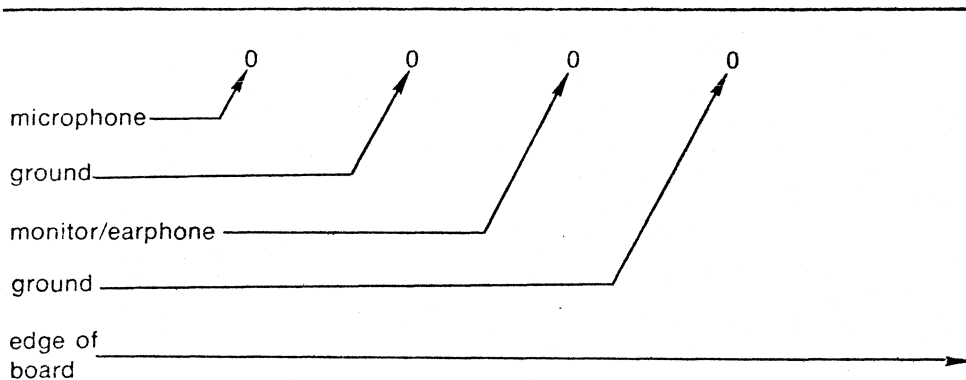


Figure 5-4. Typical Output File in Hexadecimal Format

0238

## Cassette Interface

The cassette interface is located at the rear of the SDK-2920 System Design Kit (see Figure 5-5). The lines consist of:



**Figure 5-5. Cassette Interface Hookup Locations**

0239

The following steps are required to connect a cassette:

- 1) Connect the microphone and its ground on the cassette recorder to the microphone terminals (MIC and GND) on the SDK-2920 System Design Kit and connect the cassette monitor and its ground to the earphone terminals (EAR and GND) on the kit.
- 2) Set the cassette volume level to mid point.
- 3) Verify the recorder has a cassette tape in place.
- 4) To read from cassette to SDK, use "play" mode. To write from SDK to cassette, use "record" mode; allow time at the beginning of a tape for the tape leader.



The applications side functions independently from the development side. Separate power connections are provided and space is available in the breadboard area for additional power connections. This section will introduce the 2920 and discuss the use of the applications side of the board.

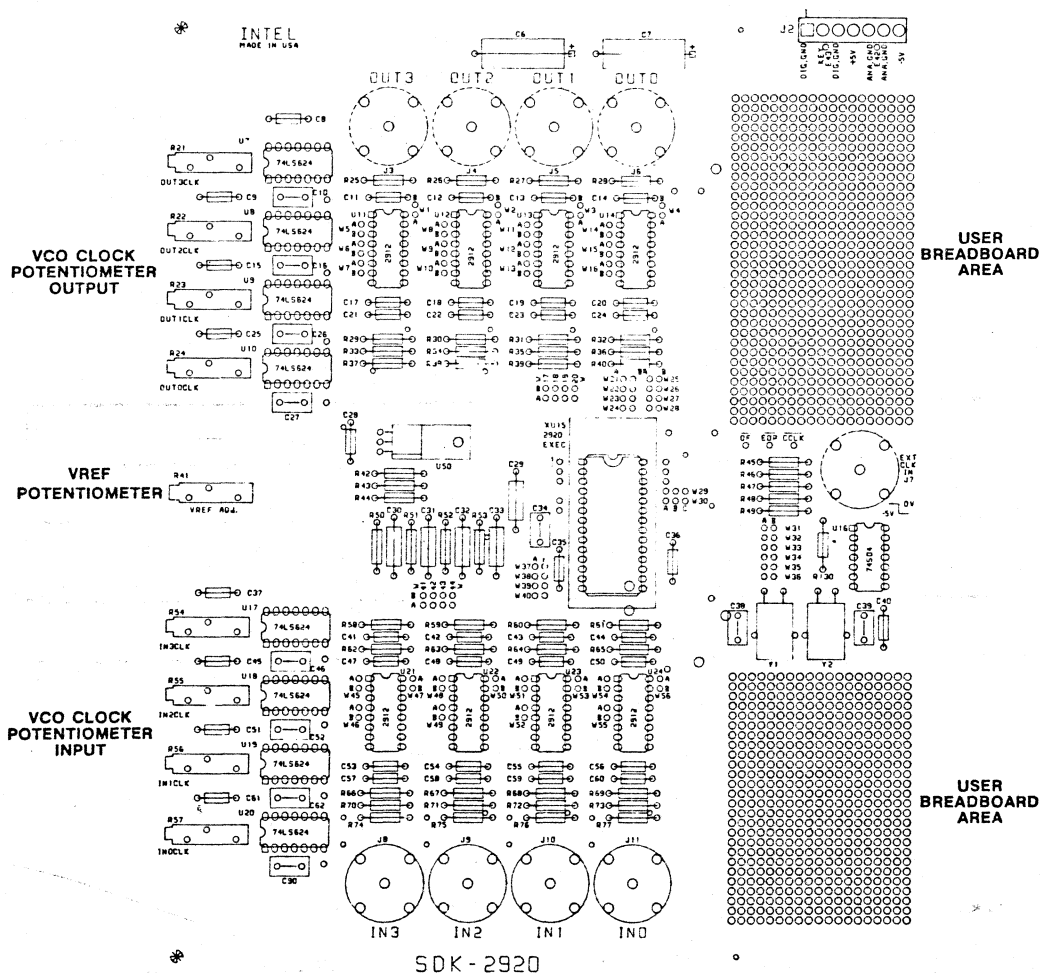


Figure 6-1. Applications Side

0240

### Example of 2920 Application

After a 2920 program has been developed and assembled, the user has two choices of program verification.

The first method uses the Intel SM2920 Simulator to simulate the execution of programs written for the 2920. The simulator requires as a minimum configuration an Inteltec development system (model 800 or Series II) with 64K RAM, TTY or CRT for console input and output, and a single diskette drive. The simulator allows you to use symbolic references for changing or displaying all 2920 registers, flags, and user-named locations in program or data memory. The trace feature enables you to see when and how selected signals are sampled and handled by your program. The SDK-2920 uploads to the simulator.

The second method of program verification is testing the programmed 2920 in your application by monitoring inputs and outputs. Component and lead positioning was carefully chosen to provide a low noise environment. This leaves you free to concentrate on your design without worrying about basic layout procedures.

## Power Supplies

You will need +5 volts and -5 volts,  $\pm 5\%$  for the 2920 signal processor and the 2912 digital filters, if used. Absolute power supply current requirements cannot be predicted since each user's application is different. The following table (Table 6-1) will assist you in determining your power needs. Consider these values as minimums and add a 20% safety margin. Remember to include other components you may add to the board.

Table 6-1. Power Requirements

	+5 volts	-5 volts
One 2920/74S04 set	60ma	120ma
One 2912/74LS324 (or 74LS624) set (multiply by up to 8)	80ma	45ma

### CAUTION

Be sure power supplies are turned off while connecting leads to the application side.

See Appendix A for power connector diagram.

The power connector J2 comes with the analog and digital grounds independent from one another. If you choose, you can strap the grounds together at J2 by locating the two holes adjacent to J2, one next to the KEY and one next to pins 5 and 6. Insert one end of a short piece of bare wire into each hole and solder it into place, thus connecting the two grounds together.

## 2920 Clock Inputs

The timing for the 2920 can be controlled by either an external clock or an external crystal. A 5.000 MHz crystal has been provided with the SDK-2920 kit. If the full



192 instructions are used in the 2920 program and the clock frequency is 5.000 MHz, then the sampling frequency will be 6.5 KHz. If an external clock is used, input a square waveform with levels from 0 to -5 volts.

**CAUTION**

An external clock with a positive polarity waveform can destroy the 74S04. The 74S04 is an inverter used as a complementary driver for the 2920 clock inputs.

A resistor location, R130, is provided adjacent to the 74S04 for a convenient place to terminate the coaxial line from the signal generator. Select a resistor close to the characteristic impedance of the coax.

The maximum allowable clock frequency is determined by the 2920 suffix. See Table 6-2.

Table 6-2. Maximum Clock Frequencies

2920-18;	5.0 MHz
2920-16;	6.67 MHz

Figure 6-2 shows the pads for connecting one of two crystals' leads or the output of a 74S04 driver to the 2920 clock input. One 5.000 MHz crystal is supplied in your kit. If you use an external frequency source to drive your 2920, be sure it is jitter-free to prevent noise being introduced into your design.

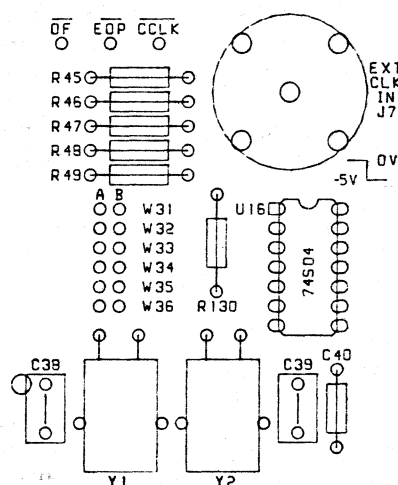


Figure 6-2. Clock Inputs

0241

### External Clock Configuration

A jumper from W31, A to W31, B and a jumper from W32, A to W32, B will connect the 74S04 clock driver to the 2920 clock inputs.

### Crystal Y1 Configuration

A jumper from W35, A to W35, B and a jumper from W36, A to W36, B will connect Y1.

### Crystal Y2 Configuration

A jumper from W33, A to W33, B and a jumper from W34, A to W34, B will connect Y2.

BE SURE THAT ONLY ONE CLOCK SOURCE IS CONFIGURED AT A TIME.

### Reference Voltage

The VREF circuit provides a low noise regulated voltage reference for the A/D and the D/A converter on the 2920. The voltage at the VREF pin on the 2920 may be set to greater than or equal to one volt or less than or equal to 2 volts. VREF determines the input and output voltage range for the 2920. When VREF equals 2 volts then the input and output voltage range is plus and minus 2 volts. If VREF is set to 1 volt then the input and output voltage range is plus and minus 1 volt. VREF is adjustable via a potentiometer located on the middle left edge of the SDK board.

### Digital Outputs

If digital outputs are going to be used, then either M1, M2 or both pins on the 2920 will have to be tied to -5 volts. The jumpers W29 and W30 configure M1 and M2. In the analog output mode, for example, M1 and M2 are both connected to +5 volts, i.e., W29, B is jumpered to W29, C and W30, B is jumpered to W30, C. Refer to the 2920 data sheet for further details on M1 and M2. When using digital outputs, pull up resistors are required for TTL levels. The application side provides traces and mounting holes for pull up resistors at W17, A and B through W24, A and B. Refer to the application side schematic for details.

OF/, EOP/, and CCLK/ are 2920 digital outputs conveniently positioned adjacent to the user's prototype area. These digital outputs have pull-up resistors provided as part of your SDK-2920 kit.

### Access To 2920 I/O Pins

If your application does not require 2912 digital filters on the input or output, your SDK-2920 kit provides user's prototype areas for the input and output of the 2920. These areas consist of groups of plated through holes on 0.1 inch centers and are convenient for bread boarding your own circuit. Use board holes W37 B through W40 B for 2920 inputs. Use board holes W25 B through W28 B for 2920 outputs (0-3). Outputs 4 through 7 can be accessed at board holes W17 A through W20 A. Keep your signal leads to the 2920 as short as possible and use coax where needed. Refer to the application side schematic for details.

See Figure 6-3 for jumper pin arrangements allowing easy user access to the 2920.

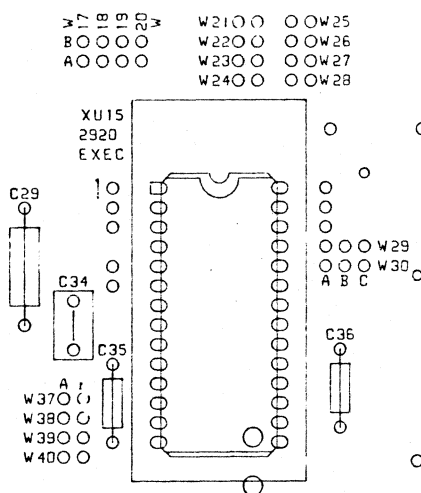


Figure 6-3. 2920 Jumper Pins

0242

## Input and Output Circuit Considerations

See Figure 6-1 for 2920 input components including BNC connectors and 2920 VC0 clock frequency adjustments.

If input signal frequencies approach the program sample rate, aliasing noise will be introduced and may become apparent at the output. The Intel 2912 filter provides a convenient controllable solution. The two 2912's supplied in your kit may be used as two input filters, two output filters, or one of each. Soldering 16 pin DIP IC sockets into the 2912 hole patterns will facilitate moving the 2912's as needed.

Each 2912 chip contains a transmit and a receive filter, as well as a power amp stage. The user has the option of setting jumpers so that various combinations of the two filters on each chip may be used. For example, to use both the transmit and receive filters with the BNC connector for IN0, connect point A to point B on the following jumpers: W37, W44, W54, W55 and W56. For other possible configurations, refer to the schematic of the Analog Execution Circuit. The advantages of different configurations are discussed in the following section.

The 2912 transmit filters have a built in 3db gain that may be utilized by shorting across resistors R70, R71, R72 and R73.

### Using the 2912 as an Anti-Aliasing and Reconstruction Filter for the 2920.

When a continuous signal is sampled, higher frequency components are generated. In addition to the original signal the sums and differences of the original signal and the harmonics of the sampling frequency are generated. Assuming an input spectrum  $F(j\omega)$  and a sampling frequency  $F_s$ , the output spectrum for square-topped sampling  $F_{ST}(j\omega)$  is found to be

$$F_{ST}(j\omega) = \frac{\tau}{T} \frac{\sin(\omega\tau/2)}{\omega\tau/2} \sum_{n=-\infty}^{\infty} F[j(\omega - n\omega_s)]$$

1.536MHz

From this equation, the gain is a continuous function of frequency defined by.

$$\frac{\tau \sin(\omega t/2)}{T \omega t/2}$$

where  $\tau$  is the sample pulse width,  $t$  is time,  $T$  the sample period, and  $\omega$  the frequency in radians per second.

A sampling theorem relating the minimum required sampling frequency to the signal bandwidth can be stated as follows: if a signal  $F(t)$ , a real function of time, is sampled instantaneously at regular intervals, at a rate higher than twice the signal bandwidth, then the sampled signal contains all the significant information of the original signal.

To ensure that there is no aliasing distortion in the 2920, the 2912 is used as an anti-aliasing filter to band limit the input signal before it is sampled. The band limited signal is sampled by the 2920 and converted to a digital signal. Once the signal is in digital form the 2920 can process it. After the 2920 has processed the signal it is converted back to analog and sampled again. The sample is held for the entire duration of the sample period. The 2912 can now be used on the output to reconstruct the quantized output of the 2920 into a smooth continuous signal.

Before one can specify the requirements for an anti-aliasing and reconstruction filter, one must first specify what distortion free dynamic range is needed. Once this is known, the frequency components generated from sampling can be attenuated, using the input and output filters, to the point where they do not distort the signal within the distortion free dynamic range of the system. For example, if the distortion free dynamic range of a 2920 application is 30 dB, then the input anti-aliasing filter should attenuate all input frequencies greater than or equal to the sampling frequency minus the bandwidth ( $F_s - BW$ ) by greater than 30 dB. (See Figure 6-4.) This way any aliasing distortion will be less than 30 dB within the signal bandwidth and will not introduce any distortion within the dynamic range of the system. For the reconstruction filter, if we assume that no aliasing distortion has been introduced within the 2920 then all of the frequency components outside of the signal bandwidth should be attenuated greater than 30 dB.

The transmit and receive filter each have about 30 dB of attenuation in the band reject region. Therefore if the distortion free dynamic range is 30 dB or less, then the transmit or receive filter of the 2912 can be used as an anti-aliasing input filter. The receive filter can be used as the output reconstruction filter. However if the distortion free dynamic range is greater than 30 dB then both the transmit and receive filter will have to be cascaded together to be used as an anti-aliasing filter, and another 2912, with both filters cascaded together, will be used for the receive filter. In this configuration the 2912 will provide 60 dB of rejection, however, the distortion free dynamic range will now be limited to 54 dB because of the 2920's quantization noise.

For a 30 dB dynamic range, the user has a choice of using the transmit filter or the receive filter. If notched frequencies are of importance to the application, then the receive filter can be used for anti-aliasing. However, the receive filter has a  $x/\sin(x)$  response. The  $x/\sin(x)$  response can be made negligible by using a one pole filter to attenuate the gain added by the  $x/\sin(x)$  equalization. The  $x/\sin(x)$  response adds a gain of about 2.5 dB at 3.3 KHz. The one pole filter can be selected to offset this gain. This feature can be added to the output of the receive filter section of the input 2912s as shown on the applications side schematic diagram. In this diagram, a 560 ohm resistor and a .068 micro farad cap are used. These values were chosen for the case when the 2912 is operating at the designed bandwidth. If another bandwidth is used, then this resistor and capacitor should be modified accordingly.

The power amp should be the final stage of the 2912 before going into the 2920. When the 2920 is sampling, the input impedance is only 1.5K ohms. The power amp

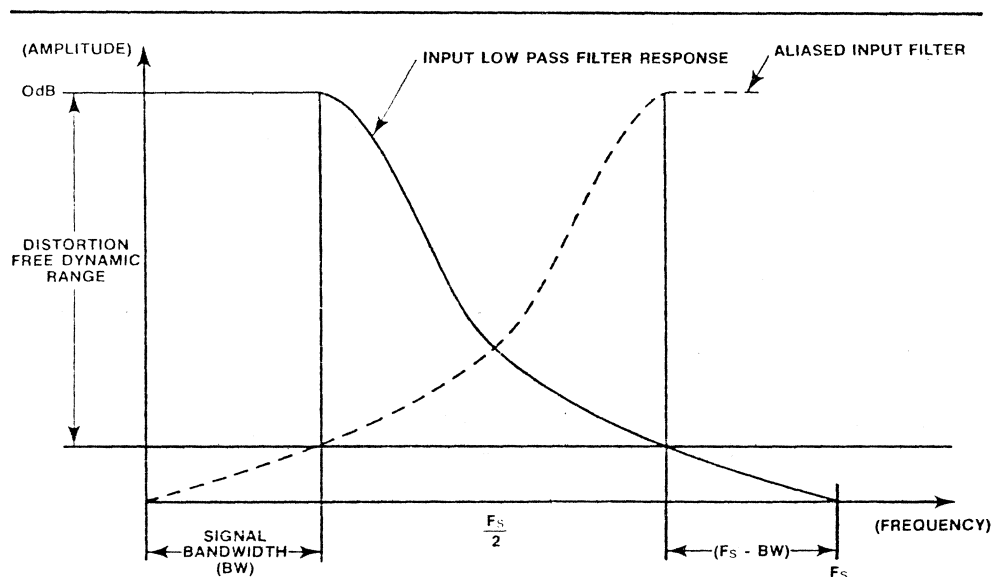


Figure 6-4. Aliasing Considerations

0243

on the 2912 is able to drive this load. The input to the 2920 can be AC coupled with a 1 micro farad cap to eliminate the output offset of the power amp. If the 1 micro farad cap is used, then the 150K ohm resistor must also be used to prevent the input signal from floating.

## Output Circuit Considerations

See Figure 6-1 for 2920 output components including BNC connectors and 2912 VCO clock frequency adjustments points.

## Sample Program

The sample program uses IN0 and OUT0 BNC connectors to produce the waveforms shown in Figure 6-5. The output signal indicated at point A in the figure has a frequency of 1.6 KHz and is generated during the negative portion of the input signal (5 MHz sine wave in the sample shown). The following steps set up the SDK for sample program operation.

1. Power up the development side of the SDK and program the 2920 with the sample program. (Remove the 2920 from the programming socket before powering up.) The command sequence is as follows:

Key	Display
<b>RESET</b>	EDIT LOAD LIST CONV
<b>LOAD</b>	2920=1 AUX=2 CASS=3
<b>1</b>	VERIFY 2920 SOCKET (CR)
<b>CR</b>	READ=1 PROG/VER=2 CMPR=3
<b>2</b>	PRESS — CR — TO PROGRAM
<b>CR</b>	2920 TRANSFER ACTIVE
then	062 = CHECKSUM (CR)
<b>RESET</b>	EDIT LOAD LIST CONV

2. Move the 2920 to the socket on the Application section.

3. Connect an input signal (oscillator) to the BNC input connector IN0. This input signal can be a sine wave, square wave, or any other waveform that goes both positive and negative (see Figure 6-4).
4. Connect an oscilloscope to BNC output connector OUT0.
5. Set the input signal frequency to 100Hz.
6. Apply power to the Applications section of the kit.
7. The oscilloscope should display 5-millisecond bursts of 1.6 kHz output (for 5 MHz input operation).

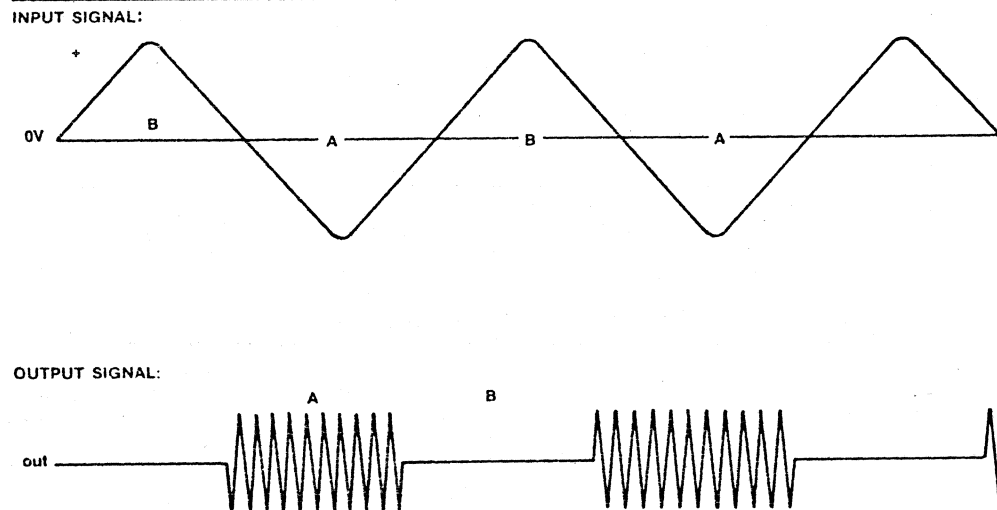


Figure 6-5. Sample Program Input and Output Waveforms

0244

### Sample Program Listing

```

0 408ACB SUB Y00, KP1, L01, NOP
1 4044EF LDA DAR, Y00, R00, NOP
2 7882CD ADD Y00, KP4, L01, CNDS
3 0066EB SUB DAR, DAR, R00, IN0
4 0000FF LDA Y01, Y00, R00, IN0
5 4882FB SUB Y01, KP4, R00, NOP
6 6008D7 ABS Y01, Y01, L01, CVTS
7 4882FB SUB Y01, KP4, R00, NOP
8 4008DD ADD Y01, Y01, L01, NOP
9 4482EF LDA Y02, KP0, R00, NOP
10 7408EF LDA Y02, Y01, R00, CNDS
11 4244EF LDA DAR, Y02, R00, NOP
12 4000EF LDA Y00, Y00, R00, NOP
... (13 through 17 are NOPs)

18 8000EF LDA Y00, Y00, R00, OUT0
... (19 through 23 are identical to 18)

24 4000EF LDA Y00, Y00, R00, NOP
... (25 through 187 are NOPs)

188 5000EF LDA Y00, Y00, R00, EOP,
189 4000EF LDA Y00, Y00, R00, NOP
... (190 through 192 are NOPs)

```

## 2920 Socketing Procedure

### CAUTION

Since the 2920 is a MOS device, care should be taken to prevent static discharge from damaging it during handling.

### CAUTION

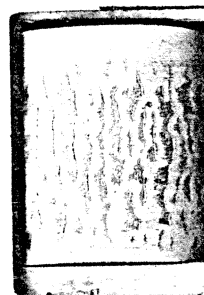
Be sure power is turned off and supplies have discharged before inserting or removing the 2920 from its socket or moving the locking handle to its "up" position.

### NOTE

The programming socket on the development side requires a different procedure from the application side socket. On the development side, the 2920 must be removed from the programming socket before powering the side up or down.

The zero insertion force socket locking handle should always be up (perpendicular to the board) before attempting to insert or remove the 2920. When inserting, 2920 pin 1 should go into the socket pin closest to the locking handle.

After the 2920 has been placed in the socket, move the locking handle down until it seats parallel to the board. Moving the locking handle to its unlocked position while power is applied may cause damage to the 2920.







## APPENDIX A POWER REQUIREMENTS

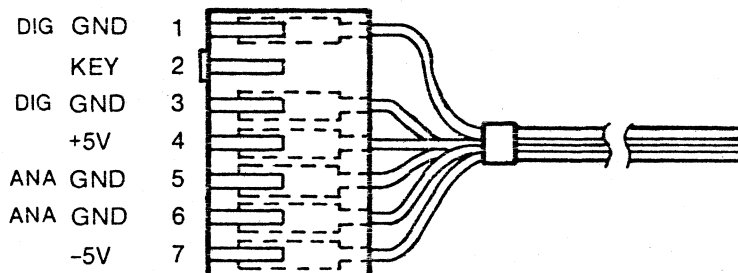
The power for development board operation and applications board operation utilizes separate connectors. The voltages needed for development are +5v and +12v. Required operating voltages for the applications board are +5v and -5v.

### DEVELOPMENT SIDE POWER REQUIREMENTS

<u>VOLTAGE</u>	<u>CURRENT</u>	<u>COMMENTS</u>
+5v $\pm$ 5%	1.0A	needed for basic operation (Vcc)
+12	100 ma	needed for basic operation (V1)
-12	100 ma	needed for RS-232 interface (Vap 1)

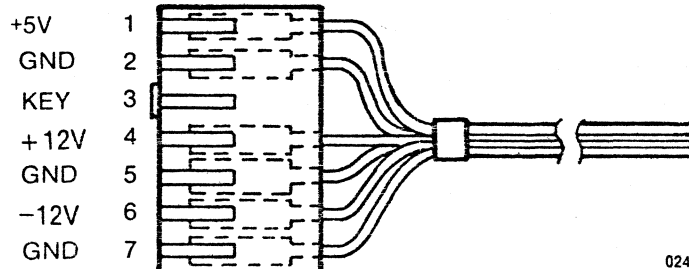
<u>VOLTAGE</u>	<u>CURRENT</u>	<u>COMMENTS</u>
+5v $\pm$ 5%	300 mA	needed for board as supplied (Vcc)
-5 $\pm$ 5%	250 mA	needed for board as supplied (Vap 2)
+5v $\pm$ 5%	200mA	needed for each 2912/74LS324 (or 74LS624) pair added (Vcc)
-5v $\pm$ 5%	200mA	needed for each 2912/74LS324 (or 74LS624) pair added (Vap 2)

Applications Side  
Power Connector  
Top View



0245

Development Side  
Power Connector  
Top View



0246

When power is applied correctly to the development side, the monitor is initiated to display the following messages:

Display

SDK-2920 MONITOR VER x.y

EDIT LOAD LIST CONV

Comment

Held on display for a short period.

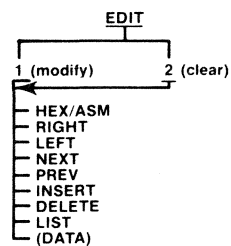
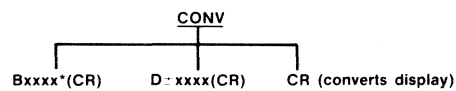
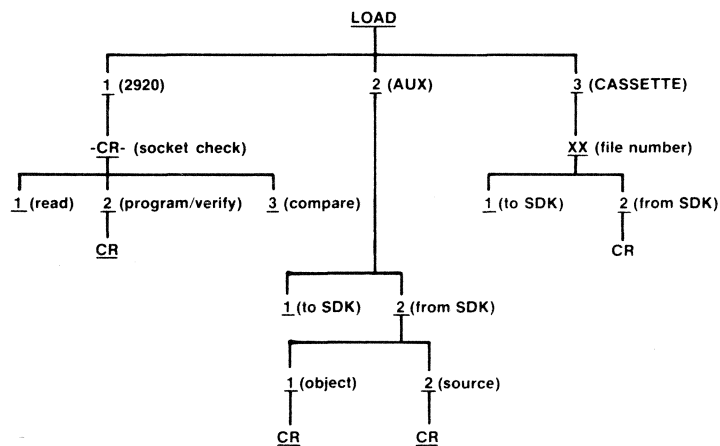
Held on display until a legal key is pressed.



Remove the 2920 from the programming socket while powering the development side up or down.



## APPENDIX B COMMAND SUMMARY



**LIST**

0247

B-1/B-2





## APPENDIX C ERROR MESSAGES

Display	Description
ILLEGAL ENTRY	Keyboard entry illegal
ERR AT LOC XXX	2920 EPROM write failure at location XXX
CHECKSUM ERROR	Cassette or development system data failed block checksum
XXX - FIELD INCORRECT	Assembler could not assemble field XXX
SYMBOL OVERFLOW	Symbol table full
NO EOP	EOP instruction not found before EPROM program
25v MISSING	2920 EPROM program logic cannot detect 25v
STUCK BIT XXX	Location XXX within the 2920 EPROM contains a bit which is programmed in the opposite direction from the program in memory.
SELECT BAUD	Place a jumper on one pair of the baud rate selection pins and press RESET.
ENTER NUMBER	Normal CONV command prompt; also appears after an illegal entry with CONV.

The following messages appear prior to programming the 2920 EPROM. After each warning is displayed, press the CR key to continue or the "0" key to bypass error checking. Warnings may not be fatal but should be understood before proceeding to program the 2920 EPROM.

Display	Description
EOP BOUNDARY WARNING	EOP instruction not on address multiple of four
WARNING	2 CNDX,SUB,DAR as destination
	3 CNDS,SUB,DAR as destination
	4 CVTX with DAR as destination
	5 CVTS with DAR as destination
	6 LIM with SHIFT
	7 CNDX with AND, LIM, or ABS
	8 CNDS with AND, LIM, or ABS
	9 OUT instruction follows DAR as destination
	10 OUT instruction follows CNDX, SUB
	11 OUT instruction follows CNDS, SUB
	12 CVTX follows DAR as destination
	13 CVTS follows DAR as destination
	14 CVTX follows CNDX, SUB
	15 CVTS follows CNDX, SUB
	16 CVTX follows CNDS, SUB
	17 CVTS follows CNDS, SUB
	18 CVTX follows CVTX
	19 CVTS follows CVTX
	20 CVTX follows CVTS
	21 CVTS follows CVTS
	22 CVTX follows IN
	23 CVTS follows IN





## APPENDIX D TWO'S COMPLEMENT DATA HANDLING IN THE 2920

Data in the 2920 are stored using a two's complement binary form. Using this form, the highest order bit indicates the sign of the value, with this bit being zero (0) for positive and zero values, and one (1) for negative values. If the intended value is positive, the remaining bits correspond to that value, independent of the sign bit. If the intended value is negative, then the remaining bits correspond to the number (one minus that value).

A convention used with the 2920 places an imaginary binary point just to the right of the highest order (25th) bit, as shown below:

1.011010111001 111110 000100

Each bit to the right of the binary point has a positive fractional weight associated with it, the first having the value  $2^{-1} = \frac{1}{2}$ , the second  $2^{-2} = \frac{1}{4}$ , and so on. If  $x$  is the number represented by the bits to the right of the binary point, then  $0 \leq x < 1.0$ . If  $s$  represents the sign bit (0 for non-negative values, 1 for negative), then the full 25-bit number represents the value  $-s + x$ .

Two's complement arithmetic is used because it allows relatively simple hardware realizations of arithmetic functions. Addition in two's complement follows normal binary addition rules, and can be realized using standard adder building blocks. If two numbers of like sign bit are added and the sign bit of the result differs from that of the original operands, the result is too large in magnitude to be contained within the allotted number of bits. In this case an "overflow" is said to have occurred.

Subtraction in two's complement arithmetic may be done by adding the two's complement of the subtrahend. The two's complement of a number is formed by first taking the one's complement and then adding a 1 in the lowest order position. The one's complement is formed by complementing all bits in place, i.e., replacing all original zeroes with ones, and all original ones with zeroes. (Note that the number  $-1.0$  has no valid two's complement in the 25-bit number system used.) In practice, subtraction is accomplished by adding the one's complement, and forcing a carry input into the lowest order adder stage — which is equivalent to adding a 1 in the lowest order position.

Using two's complement arithmetic therefore simplifies addition and subtraction as compared with sign/magnitude representation in that no sign bit testing of either operand is necessary to set up for addition or subtraction. Only one set of adders is needed because the conversion from one's complement to two's can be achieved within the adder.

Multiplication and division by powers of two corresponds to shifts left or right respectively. When shifting left, the low order bit is filled with zeroes and when shifting right, the high order bit is filled with the sign bit. To extend precision to the left, the sign bit is extended into each added position before any shift operations are done. The sign bit behaves as if it extends to the left on to infinity. Overflow corresponds to the case where the recorded sign bit does not correspond to the sign bit at infinity.

In the 2920, arithmetic is performed with a left extension to a total of 28 bits, adequate to perform any 2920 operation without possibility of overflow. Thus the highest order bit corresponds to the sign bit at infinity. If the storable portion of the result (low 25-bits) does not correspond to the correct result, an overflow is indicated, and if overflow limiting is enabled, the 25-bit value stored is the positive extremum (if the correct sign bit was 0) or the negative extremum (if the correct sign bit was 1).

positive extremum = 0.1111 1111 1111 1111 1111 1111 = approx. + 1.0  
 negative extremum = 1.0000 0000 0000 0000 0000 0000 = -1.0

In two's complement arithmetic, multiplication can be performed in a manner similar to that used for positive binary numbers. However, because the sign bit has a negative rather than a positive weight, some additional corrections are needed.

Multiplication in the 2920 may be achieved using the conditional add, with the multiplier being loaded into the DAR, and the multiplicand being conditionally added to the (partial) product. Because adds in the 2920 provide for sign extension during shifting, a positive multiplier can produce a correct product without any further correction, shown in the examples below.

$$\begin{array}{r}
 1111.11 \text{ } (-\frac{1}{4}) \\
 \times 0000.11 \text{ } (+\frac{3}{4}) \\
 \hline
 \dots 1111.111 \\
 \dots 1111.1111 \\
 \hline
 \dots 11111.1101 = -1 + 13/16 = -3/16
 \end{array}$$

Note that in each case, the sign bit was extended to the left in the partial products. The example shown above is drawn in a manner different from that used in grade school arithmetic classes. The somewhat different display results from noting that each bit of the multiplier to the right of the binary point has a weight equal to some negative power of 2, i.e., is equivalent to a right shift of one or more positions.

$$\begin{array}{r}
 0.1100 \text{ } (3/4) \\
 \times 0.1101 \text{ } (13/16) \\
 \hline
 0000.01100 \\
 0000.001100 \\
 0000.0000000 \\
 0000.00001101 \\
 \hline
 00000.10011101
 \end{array}$$

Thus if the multiplication is done starting at the binary point of the multiplier, and running through the multiplier from left to right, the binary point can be maintained (and aligned) for each partial product. Each bit of the multiplier corresponds to a possible addition of the multiplicand, shifted to the right by one or more positions, to the product. If the multiplier bit is a 1, the addition takes place, otherwise it does not take place.

If the sign bit of the multiplier is non-zero, because this bit has a negative-weight the multiplicand should be subtracted from the product. In the 2920 this function is achieved by complementing the multiplicand, and conditionally adding the resulting complement to the product based on the sign bit of the multiplier.

Division tends to be complex in two's complement arithmetic, and so may be simplified by extracting the signs of the operands, performing the division using only the magnitudes of the dividend and divisor. The quotient is converted to the proper sign based on the extracted signs.

Restoring binary division is performed using a series of test subtractions of the divisor from the dividend, with the original value restored if the result becomes negative. The sequence of test subtractions proceeds from left to right, with each successful subtraction (one leaving a positive difference, thus not requiring restoration) reducing the magnitude of the dividend. The locations of the successful subtractions are noted by ones, those unsuccessful by zeroes, in the DAR.

Restoration of the dividend corresponds to adding the divisor back to the dividend. Because this operation is followed by a test subtraction with the divisor shifted one



position further right, the restoration/test subtraction sequence can be replaced by a single addition of the divisor after it is shifted to the right. (As a right shift is equivalent to a multiply by  $1/2$ , the first sequence is  $+d-d/2 = d/2$ ; the second operation is  $+d/2$ .)

In the 2920, the conditional subtract operation is used to perform this non-restoring divided algorithm. For any arithmetic operation, the high order from the extended arithmetic is saved for possible testing by the conditional subtract instruction. This carry has the same value as the sign of the result generating it, i.e., 1 for negative, 0 for non-negative numbers. The conditional subtract performs the addition or subtraction required by the previous result (i.e., carry), and then stores the new result of the operation in a designated location (bit) of the DAR as selected by the condition code used.





## APPENDIX E DISCUSSION OF CARRY AND OVERFLOW CONDITIONS

The detailed ramifications of carry and overflow are discussed in this appendix. The tables show the three major forms and the possible cases in each.

Overflow occurs when ALU operations produce numbers outside the legal range of  $-1.0 \leq x < +1.0$ .

Normal standard carry logic applies to the ALU instructions ADD, SUB, ABA, LIM, AND. For the instructions XOR, ABS, and LDA, the carry logic includes additional considerations.

### Normal Carry and Overflow

The 2920 standard representation of data is a signed 25-bit binary fraction. Positive data can be considered simply 24-bit fractions with a sign bit, e.g., 0.100000000000000000000000 means  $+1/2$ . Negative data have a sign bit of 1 with the remaining 24 bits representing the two's complement of the value, i.e., one minus that value. An example:

1.010000000000000000000000 means  $-3/4$ .

However, the capacity to shift left two positions makes it necessary to allow for a 26th and 27th bit for the sign. A 28th bit is necessary to preserve the sign in the case of carry information if two numbers are left-shifted and then added.

Therefore, the 2920 logic carries 28 bits, four bits to the left of the imaginary binary point and 24 bits to its right:

ssss.bbbb bbbb bbbb bbbb bbbb

If the source operand is to be negated during an instruction, then before the indicated operation is carried out, the one's complement of the source is formed by complementing all its bits and setting the carry-in bit to one. This happens in three circumstances: in taking the absolute value of a negative number, in an unconditional subtraction, or in a conditional subtraction when the prior carry was 0.

Standard carry, then, is propagated to the left, beginning at the least-significant (right-most) bit and continuing into the sign bits if necessary. Carry into the sign bits may mean an overflow condition, since in overflow the four sign bits become unequal. The leftmost bit of the source operand always retains the original sign even if shifting occurs.

Normal practice is to keep the numbers scaled between  $-1.0$  and  $+1.0$ , such that the arithmetic operations do not create values outside this range. If out-of-range values do result, this is an overflow situation.

Conditional I/O codes do not affect carry and overflow for standard carry instructions (except conditional subtract). The calculation acts as if a straight add were being done: if a carry into the sign bit occurs, then the carry flag is set. (Subtraction is performed as an add after taking the two's complement of the source operand and setting carry-in to 1.)

LIM produces a  $+1.0$  or a  $-1.0$  using the sign of the source only, and sets the carry to 0. Overflow for LIM depends on whether a left-shift occurred. When overflow limiting is enabled and an overflow condition occurs on an ADD, ABS, or ABA instruction, the result is limited, i.e., becomes  $-1.0$  or  $+1.0$  (This never applies to AND.)

The source operand is also a six-bit address pointing into RAM. Each address bit is located as follows:

ADDR BIT	0(LSB)	A0
	1	A1
	2	A2
	3	A3
	4	A4
	5(MSB)	A5

The source operand constant multipliers have opcodes as follows:

KP7	110111	KM1	111111
KP6	110110	KM2	111110
KP5	110101	KM3	111101
KP4	110100	KM4	111100
KP3	110011	KM5	111011
KP2	110010	KM6	111010
KP1	110001	KM7	111001
KP0	110000	KM8	111000

### Shift Code Field

S3	S2	S1	S0	Mnemonic
1	1	0	0	R13
1	0	1	1	R12
1	0	1	0	R11
1	0	0	1	R10
1	0	0	0	R09
0	1	1	1	R08
0	1	1	0	R07
0	1	0	1	R06
0	1	0	0	R05
0	0	1	1	R04
0	0	1	0	R03
0	0	0	1	R02
0	0	0	0	R01
1	1	0	1	L01
1	1	1	0	L02
1	1	1	1	R00

## Input/Output Code Field

The field is located in bits 23-19 and is encoded in the following manner:

MSB		LSB			Mnemonic
ADF1	ADF0	ADK2	ADK1	ADK0	
0	0	0	0	0	IN0
0	0	0	0	1	IN1
0	0	0	1	0	IN2
0	0	0	1	1	IN3
0	0	1	0	0	NOP
0	0	1	0	1	EOP
0	0	1	1	0	CVTS
0	0	1	1	1	CNDS
0	1	0	0	0	OUT0
			•		•
			•		•
			•		•
0	1	1	1	1	OUT7
1	0	0	0	0	CVT0
			•		•
			•		•
			•		•
1	0	1	1	1	CVT7
1	1	0	0	0	CND0
			•		•
			•		•
			•		•
1	1	1	1	1	CND7





# APPENDIX G HEXADECIMAL/BINARY CONVERSION TABLE

## POWERS OF 16 (IN BASE 10)

				16 <sup>n</sup>	n	16 <sup>-n</sup>					
				1	0	1.00000	00000	00000	00000	x 10 <sup>0</sup>	
				16	1	0.62500	00000	00000	00000	x 10 <sup>-1</sup>	
				256	2	0.39062	50000	0000	0000	x 10 <sup>-2</sup>	
				4	096	3	0.24414	06250	00000	00000 x 10 <sup>-3</sup>	
				65	536	4	0.15258	78906	25000	00000 x 10 <sup>-4</sup>	
				1	048	576	5	0.95367	43164	06250 00000 x 10 <sup>-6</sup>	
				16	777	216	6	0.59604	64477	53906 25000 x 10 <sup>-8</sup>	
				268	435	456	7	0.37252	90298	46191 40625 x 10 <sup>-8</sup>	
				4	294	967	296	8	0.23283	06436 53869 62891 x 10 <sup>-9</sup>	
				68	719	476	736	9	0.14551	91522 83668 51807 x 10 <sup>-10</sup>	
				1	099	511	627	776	10	0.90949 47017 72928 23792 x 10 <sup>-12</sup>	
				17	592	186	044	416	11	0.56843 41886 08080 14870 x 10 <sup>-13</sup>	
				281	474	976	710	656	12	0.35527 13678 80050 09294 x 10 <sup>-14</sup>	
				4	503	599	627	370	496	13	0.22204 46049 25031 30808 x 10 <sup>-15</sup>
				72	057	594	037	927	936	14	0.13877 78780 78144 56755 x 10 <sup>-16</sup>
1	152	921	504	606	846	976	15	0.86736	17379	88403	54721 x 10 <sup>-18</sup>

## POWERS OF 10 (IN BASE 16)

				10 <sup>n</sup>	n	10 <sup>-n</sup>				
				1	0	1.0000	0000	0000	0000	× 16 <sup>0</sup>
				A	1	0.1999	9999	9999	999A	× 16 <sup>0</sup>
				64	2	0.2BF5	C2BF	5C28	F5C3	× 16 <sup>-1</sup>
				3E8	3	0.4189	374B	C6A7	EF9E	× 16 <sup>-2</sup>
				2710	4	0.68DB	8BAC	710C	B296	× 16 <sup>-3</sup>
			1	86A0	5	0.A7C5	AC47	1B47	8423	× 16 <sup>-4</sup>
			F	4240	6	0.10C6	F7A0	B5ED	8D37	× 16 <sup>-4</sup>
			98	9680	7	0.1AD7	F29A	BCAF	4858	× 16 <sup>-5</sup>
			5F5	E100	8	0.2AF3	1DC4	6118	73BF	× 16 <sup>-6</sup>
			3B9A	CA00	9	0.44B8	2FA0	9B5A	52CC	× 16 <sup>-7</sup>
		2	540B	E400	10	0.6DF3	7F67	SEF6	EADF	× 16 <sup>-8</sup>
		17	4876	EB00	11	0.AFEB	FF0B	CB24	AAFF	× 16 <sup>-9</sup>
		EB	D4A5	1000	12	0.1197	9981	2DEA	1119	× 16 <sup>-9</sup>
		918	4E72	AD00	13	0.1C25	C268	4976	81C2	× 16 <sup>-10</sup>
		5AF3	107A	4000	14	0.2D09	370D	4257	3604	× 16 <sup>-11</sup>
	3	BD7E	A4C6	8000	15	0.4B0E	BE7B	9D58	566D	× 16 <sup>-12</sup>
	23	8652	6FC1	0000	16	0.734A	CA5F	6226	F0AE	× 16 <sup>-13</sup>
	163	457B	5D8A	0000	17	0.B877	AA32	36A4	B449	× 16 <sup>-14</sup>
DE0	B6B3	A764	0000	18	0.1272	5DD1	D243	ABA1	× 16 <sup>-15</sup>	





**COMMAND INDEX****PAGE**

Clear Command .....	3-3
Compare 2920 EPROM .....	3-13
Convert Binary to Decimal .....	3-19
Convert Decimal to Binary .....	3-20
Hex Edit .....	3-7
LIST .....	3-10
Loading Programs .....	3-9
Modify Command .....	3-4
Program/Verify 2920 EPROM .....	3-12
Read Object Data from Cassette .....	3-17
Read Object Data from Development System .....	3-14
Read 2920 EPROM .....	3-11
RESET .....	3-3
Symbolic Edit .....	3-4
Transfer Object Data to Development System .....	3-15
Transfer Source Data to Development System .....	3-16
Transmit Object Data to Cassette .....	3-18

**TEXT INDEX****PAGE**

Baud Rate Selection .....	5-2
Carry .....	E-1
Command Summary .....	B-1
Conversion Tables .....	G-1
EOP .....	4-7
Editing .....	3-4
Error Messages .....	C-1
Instruction Memory .....	3-2
Object Code Output Format .....	5-4
Overflow .....	E-1
Sample Program .....	6-7
Serial Interface .....	5-1
Simulator .....	6-2
Specifications .....	1-2
Power Requirements .....	A-1
Two's Complement Data Handling .....	D-1
Uses for SDK-2920 Design Kit .....	1-1

